NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SYSTEM BEHAVIOR MODELS: A SURVEY OF APPROACHES

by

Scott R. Ruppel

June 2016

Thesis Advisor: Kristin Giammarco
Second Reader: John M. Green

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2016	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
SYSTEM BEHAVIOR MODELS: A SURVEY OF APPROACHES

5. FUNDING NUMBERS

6. AUTHOR(S) Scott R. Ruppel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Human designed systems are growing in complexity, with increasing numbers of components and behavior combinations, resulting in more emergent and unintended behaviors evident in operations. This thesis explores various behavior modeling approaches and their potential for exposing emergent behaviors, highlighting trends and modeling approaches. The report defines key concepts and provides a context for a comparative analysis of approaches. In particular, this report assesses a relatively new approach to behavior and architecture modeling, Monterey Phoenix (MP), and compares it with Petri nets, a well-established method. The comparison involves a simple communication process between two components, which is modeled and compared to an equivalent Petri net model. Shared outcomes involve a successful communication between the components and failure modes of the components not receiving or processing data. The models produce identical state space results. The combined state space graph of the Petri model allowed a quick assessment of all potential states but was more cumbersome to build than the MP model. A comparison of approaches charts the modeling methods against the key concepts, revealing the differences among methods, contrasted with the aspects of MP.

14. SUBJECT TERMS
Monterey Phoenix, Petri nets, behavior modeling, model-based systems engineering, modeling approaches, modeling survey

15. NUMBER OF PAGES
85

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SYSTEM BEHAVIOR MODELS: A SURVEY OF APPROACHES

Scott R. Ruppel
Civilian, Department of the Navy
B.S., University of Portland, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
June 2016

Author: Scott R. Ruppel

Approved by: Kristin Giammarco, Ph.D.
Thesis Advisor

John M. Green
Second Reader

Ronald Giachetti, Ph.D.
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Human designed systems are growing in complexity, with increasing numbers of components and behavior combinations, resulting in more emergent and unintended behaviors evident in operations. This thesis explores various behavior modeling approaches and their potential for exposing emergent behaviors, highlighting trends and modeling approaches. The report defines key concepts and provides a context for a comparative analysis of approaches. In particular, this report assesses a relatively new approach to behavior and architecture modeling, Monterey Phoenix (MP), and compares it with Petri nets, a well-established method. The comparison involves a simple communication process between two components, which is modeled and compared to an equivalent Petri net model. Shared outcomes involve a successful communication between the components and failure modes of the components not receiving or processing data. The models produce identical state space results. The combined state space graph of the Petri model allowed a quick assessment of all potential states but was more cumbersome to build than the MP model. A comparison of approaches charts the modeling methods against the key concepts, revealing the differences among methods, contrasted with the aspects of MP.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
A. PURPOSE....1
B. RESEARCH QUESTION....1
C. RESEARCH METHODOLOGY....2
D. SCOPE....3
E. STRUCTURE....3

II. CONCEPT OVERVIEWS....5
A. INTRODUCTION....5
B. MODEL-BASED ENGINEERING....5
1. Model-Driven Engineering....6
2. Model-Driven Architecture....6
C. CONCEPTS....7
1. Frameworks....7
2. Behavior Modeling and Emergent Behavior....8
3. Abstraction....9
4. Separation of Concerns....10
5. Stepwise Refinement....10
6. Formal Methods and the Small Scope Hypothesis....10
D. SUMMARY....11

III. BEHAVIOR MODELING APPROACHES....13
A. INTRODUCTION....13
2. Survey of Self-Adaptive Systems and Formal Methods....14
C. DOMAIN-SPECIFIC APPROACHES....18
1. Automotive Requirements Modeling....18
2. Automotive Control Modeling....20
D. GENERAL APPROACHES....22
1. Systems Modeling Language....22
2. System Dynamics Models....24
3. Monterey Phoenix....25
4. Agent-Based Modeling....28
5. Petri Nets....29
E. SUMMARY....30

IV. PETRI NETS COMPARED AND CONTRASTED WITH MP....33
A. INTRODUCTION....33

B. EXAMPLE MODEL—MP COMPARED WITH PETRI NETS.....33
1. MP Example.....33
2. MP Example Results.....35
3. Petri Net Example.....36
4. Results and Comparison with MP Example.....39
C. SUMMARY.....44

V. CONCLUSIONS AND FUTURE RESEARCH.....45
A. INTRODUCTION.....45
B. RESEARCH QUESTION.....45
C. CONCLUSIONS.....45
D. LIMITATIONS OF RESEARCH.....48
E. FUTURE RESEARCH.....49

APPENDIX. MP MODEL RESULTS.....51

LIST OF REFERENCES.....59

INITIAL DISTRIBUTION LIST.....65

LIST OF FIGURES

Figure 1. Spiral Model....3

Figure 2. Approaches in Modeling Scaled by Level of Abstraction. Source: (Borshchev) 2004....6

Figure 3. Projects Using Formal Methods by Domain Source: Woodcock et al. (2009)....14

Figure 4. Formal Methods in Self-Adaptive Systems. Source: (Weyns et al.) 2012....15

Figure 5. Specification Approaches Source: (Weyns et al.) 2012....15

Figure 6. Formal Verification Properties. Source: Weyns et al. (2012)....16

Figure 7. State Space of Behaviors to Properties, Numbered References. Source: Weyns et al. (2012)....17

Figure 8. Proposed Automotive Architecture Framework. Source: Yu et al. (2015)....19

Figure 9. Automotive Control System Utilizing Agents Source: Sengstacken, DeLaurentis, and Akbarzadeh-T (2007)....21

Figure 10. SysML Diagram Hierarchy. Source OMG (2015)....23

Figure 11. Activity Diagram....23

Figure 12. System Dynamics Process. Source: Forrester (1993)....24

Figure 13. Example SD Model. Source: Sterman (2001)....25

Figure 14. Event Trace A:B C (note that MP Analyzer notation uses dashed for inclusion, and solid for precedence) Source: Auguston (2009)....27

Figure 15. Example Event Trace Output....36

Figure 16. Petri Net Data Transfer Example (Open Loop)....38

Figure 17. Petri Net Data Transfer Example (Closed Loop)....38

Figure 18. Reachability Output (Open Loop)....39

Figure 19. Reachability Output (Closed Loop)....40

Figure 20. Petri Net Transition Example....41

Figure 21. Classification Results Closed Loop Model 43
Figure 22. Classification Results Open Loop Model 43

LIST OF TABLES

Table 1. Example MP Operators. Adapted from NPS (2015) 27

Table 2. Select Approaches Related to Concepts 46

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AADL	analysis and design language
ABM	agent-based modeling
ASE	International Conference on Automated Software Engineering
BPM	business process modeling
CPN	colored Petri nets
DEAS	design and evolution of autonomic application
DoDAF	Department of Defense Architecture Framework
EBM	events-based modeling
FEAF	Federal Enterprise Architecture Framework
FMEA	failure modes and effects analysis
FSE	Foundations of Software Engineering
GUI	graphical user interface
ICAC	International Conference on Autonomic Computing
ICSE	International Conference on Software Engineering
JSS	Journal of Systems and Software
MBE	model-based engineering
MBR	model-based reasoning
MBSE	model-based systems engineering
MDA	model-driven architecture
MDE	model-driven engineering
MP	Monterey Phoenix
OEM	original equipment manufacturer
PIPE	Platform Independent Petri net Editor
R&M	Reliability and Maintainability
SASO	Self-Adaptive and Self-Organizing Systems
SAVI	System Architecture Virtual Integration
SD	system dynamics
SE	systems engineering
SEAMS	Software Engineering for Adaptive and Self-Managing Systems

SysML	Systems Modeling Language
SoS	system of systems
TAAS	Transactions on Autonomous and Adaptive Systems
TEAF	Treasury Enterprise Architecture Framework
TOGAF	The Open Group Architectural Framework
TSE	Transactions on Software Engineering
UML	Unified Modeling Language
WICSA	Working International Conference on Software Architecture
WOSS	Workshop on Self-Healing

EXECUTIVE SUMMARY

Monterey Phoenix (MP) is a recent behavioral modeling framework that seeks to advance the development of formal system architecture specifications. Some of the foundational concepts that MP relies on are frameworks, abstraction, separation of concerns, stepwise refinement, small-scope hypothesis, and the use of formal methods.

To relate MP to other current Model-Based Systems Engineering (MBSE) approaches, this thesis reviews selected modeling approaches, summarizing relevant trends in the context of MP concepts. Of note, a novel automotive MBSE framework proposed by Yu et al. (2015) uses generic behavior models as a central approach for linking supplier specifications to the system specification. Additionally, two modeling surveys were reviewed, showing adaptation rates of MBSE approaches.

One approach, behavior modeling with Petri nets, was selected for experimentation. A simple communications model (two entities passing information to each other) was coded and executed in the MP Analyzer environment, producing the possible behaviors (results) of the system as event traces. The communications model also was translated into an equivalent Petri net model, to compare Petri net with MP. The Petri net model was simulated in the PIPE2 program (a popular editor), producing a state space equivalent to the MP results. In this limited example, results showed that MP and Petri nets could produce equivalent possible state spaces.

Following this, the selected approaches were compared with the concepts central to MP. The primary conclusions follow:

1. MP makes use of the concepts utilized by MP as any other method or framework reviewed. These concepts include: frameworks, behavior modeling, abstraction, separation of concerns, stepwise refinement, formal methods and the small scope hypothesis

Of the publications reviewed, no other formal, executable approach claims to search exhaustively for all possible scenarios (within a given scope) while also supporting event attributes, assertion checking, and different viewpoints.

The Virtual Integration concept described by Yu et al. could benefit from the use of MP.

> None of the approaches researched fully support all of the concepts (without the use of extensions or multiple specifications/approaches) reviewed in this thesis as well as MP does, specifically, frameworks, abstraction, behavior modeling, stepwise refinement and formal methods.

Additionally, this research determined that none of the cited research focused on the application of modeling to supportability or life cycle costs.

Notable limitations in this research include the use of a simplistic model in the Petri net and MP experimental comparison, the preliminary nature of many of the topics covered, and a limited body of research on MP available to date. Recommended future research areas span use-case specific experimentation with MP, the expansion of MP tools and features, and the further exploration of MP limitations.

LIST OF REFERENCES

Yu, Huafeng, Prachi Joshi, Jean-Pierre Talpin, Sandeep Shukla, and Shinichi Shiraishi. 2015. "The Challenge of Interoperability: Model-Based Integration for Automotive Control Software." *Proceedings of the 52nd Annual Design Automation Conference*: 58.

ACKNOWLEDGMENTS

A debt of gratitude is owed to the Systems Engineering division of Naval Air Systems Command, which graciously sponsored my participation in the Systems Engineering Management program (SEM-PD 21). I would also like to thank my advisor, Kristin Giammarco, for her support, passion for architecture modeling and optimism. I send love to my family for instilling in me at an early age a passion for literature and science. The gift of David Macaulay's book *The Way Things Work* might be one of the most pivotal points in my youth. Finally, I thank Elisa for affording time to the pursuit of this research. Her moral support and understanding made this possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> Right requires justification, wrong requires conviction. If you stay on the path of right, the guideposts are many, and you never stray. But there is no one path for wrong, no lit way, no signposts, no guide.
>
> —Gary Langford
> *Engineering Systems Integration*, 2012

A. PURPOSE

Often asserted in the multi-disciplinary field of behavior modeling are the problems associated with capturing or exposing emergent behavior, unintended or undesired interactions. The primary purpose of this thesis is to perform a literature review of Model-Driven Architecture (MDA) and Model-Based Engineering (MBE) approaches, focusing on methods and practices pertaining to exposing unwanted, unneeded or undesired interactions. A relatively new approach to formalized software system architecture specification, primarily concerned with behavior modeling in light of these issues, is proposed in Monterey Phoenix (MP) (Auguston 2009). As a secondary purpose, a simple case study of a communication process between two components is modeled in MP and compared with an equivalent model utilizing an established behavioral method, Petri nets.

B. RESEARCH QUESTION

- Primary research question: How does the Monterey Phoenix behavior modeling approach compare with other approaches that claim to expose unintended, unneeded, or undesired system interactions?

The goal of this research is to put MP into the context of existing behavior modeling approaches. The primary research question is addressed by surveying publications relevant to the key concepts of MP. A multidisciplinary literature review surveys articles, journals, books and conference proceedings that advance methods congruent with the goals and methods of MP. From the survey and MP papers, an overview of key MBE concepts and characteristics provides an overall context for the

thesis. To relate MP to other approaches, this thesis reviews a selection of behavior modeling methods representing a sampling of the general state of behavior modeling approaches. Following this, this research contrasts one of the approaches (Petri nets) with MP via experimentation with a simple behavior model.

C. RESEARCH METHODOLOGY

This research is multi-disciplinary, covering a diversity of topics within systems and software engineering and multiple domain applications. As such, multiple academic databases (general and domain-specific) were employed. Sources were limited to those published within the last two decades to maximize the maturity of sources and minimize ambiguity in conclusions. However, certain fundamental concepts date back to the 1970s, such as the idea of abstract data types (Wulf 1980). The broadness in topic areas and the varying maturity of source material necessitated extensive review. Of the 115 items reviewed, this report cites approximately 75 articles. It excludes sources with little to no connection to either a general form of modeling or a direct claim of exposing latent behavior. Emphasis was placed on research that shared common characteristics and concepts employed by MP, further detailed in Chapter II.

A spiral model was chosen for researching and structuring this thesis, shown in Figure 1. This approach allowed multiple iterations of source material review (extraction of key concepts), synthesis (documenting concisely), and analysis (drawing conclusions) of interim results. The goal of this approach is the production of a set of clear conclusions relevant to the behavior modeling approach comparisons by integrating relevant concepts and applications and refining through iteration.

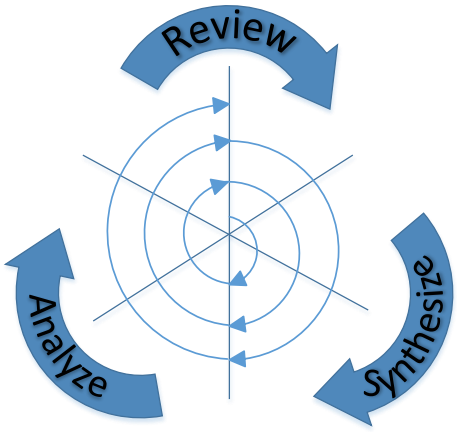


Figure 1. Spiral Model

D. SCOPE

The research is limited to a literature review, limited experimentation, and a comparison of methods and approaches pertinent to the concepts employed by MP. A possible limiting factor in conducting this research is that MP is maturing. This research is limited to the current state of MP at the time of this writing.

E. STRUCTURE

Chapter I introduces the purpose, the research question, methodology, scope, and structure. Chapter II provides general background and context for the primary research question, and a discussion of general modeling concepts relevant throughout this thesis. To provide a general understanding of relevant modeling concepts, topics such as frameworks, abstraction, and separation of concerns are described. Chapter III presents selected surveys highlighting trends and adoption rates of formal and self-adaptive systems modeling methods. It introduces selected behavior modeling approaches, focusing specifically on SysML, System Dynamics, MP, ABM and Petri nets. Chapter IV explores the relationship of MP to Petri nets by comparing models experimentally. It provides a summary of the design of each model, the experimentation process and the results. Chapter V provides primary conclusions, limitations, and recommendations for

further research. The research question is revisited, and the selected modeling approaches are compared against the concepts from Chapter II.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CONCEPT OVERVIEWS

A. INTRODUCTION

The following introduces a summary of key concepts and characteristics that underpin MP as a basis for the research supporting the thesis. The concepts are later cross-referenced in the literature review and compared with selected approaches.

B. MODEL-BASED ENGINEERING

Model-based engineering, model-based systems engineering (MBSE), and model and simulation-based Engineering (M&SBE) all refer to the use of models and simulations versus traditional document-based engineering artifacts (Stefan 2007). There are numerous and growing approaches, methodologies and frameworks. The Unified Modeling Language (UML) and Systems Modeling Language (SysML) are popular languages. The Department of Defense Architecture Framework (DoDAF) is a popular architecture framework, which describes the visualization of the different stakeholder, operational, and systems viewpoints and models (Giammarco 2007). One aspect each framework shares, primarily concerned with interaction, is behavior modeling.

Behavior modeling approaches can be differentiated by the level of abstraction supported and whether they are mainly continuous or discrete (Borshchev 2004). Figure 2 shows general categories and characteristics, separated by level of abstraction and time (discrete or continuous). These are discussed in more detail in Chapter III. The following two sections describe subsets of MBE.

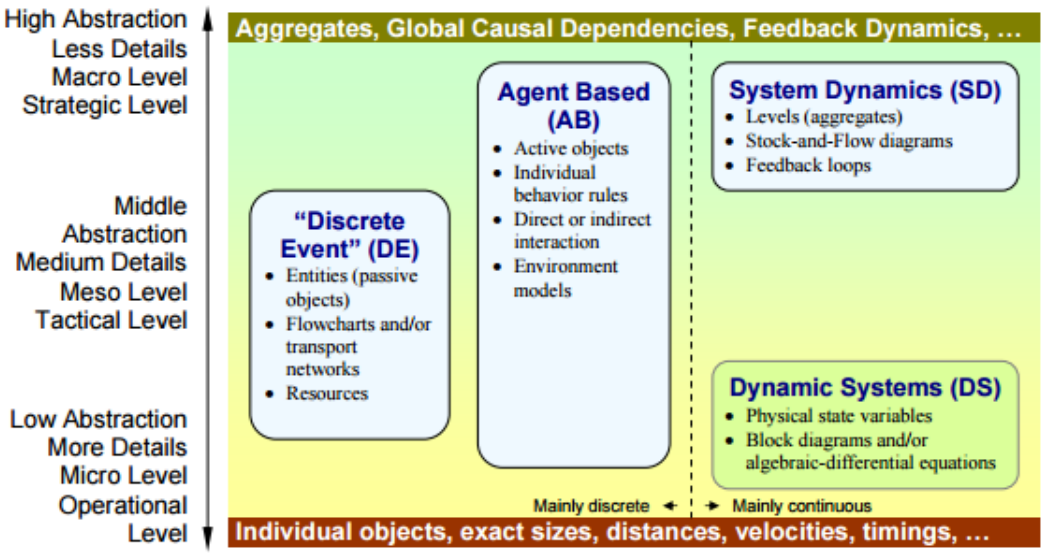


Figure 2. Approaches in Modeling Scaled by Level of Abstraction. Source: (Borshchev) 2004

1. Model-Driven Engineering

Model-driven engineering (MDE) is a subset of MBE, distinguished by a structured and coordinated use of models and simulations in the engineering (or in the software engineering) development process. A major motivation for adoption of MDE is to provide the means of dealing with the increased complexity of many facets of modern development and research (Hutchinson 2011). The F-35 Joint Strike Fighter is on such example of a complex development program utilizing MDE, with an estimated 24 million lines of code (in comparison to the F-22's 1.7 million lines of code (Hagen and Sorenson 2013). In the context of software, MDE records the organization and mapping of models by combining "process and analysis with architecture" (Kent 2002, 286). The use of MDE can aid in verification, integration and interoperability.

2. Model-Driven Architecture

Model-Driven Architecture (MDA) complements and facilitates MDE, providing guiding principles for applying MDE (Bézivin 2004). MDA provides standards, toolsets, and defined frameworks are guiding and promoting interoperability and allow executable models (Mellor et al. 2004). "MDA advocates modeling systems from three viewpoints:

computation independent, platform independent, and platform specific viewpoints" (France and Rumpe 2007, 44). Multiple viewpoints allow tailoring to a stakeholder's specific use case, decoupling tasks that are not needed for a specific user. Architectures contain a language, which defines the semantics that can be utilized and the rulesets that apply. Modeling languages have a vocabulary (components of a model) and grammar (how they relate) (Maier 2009). This grammar, or code, can then be automated, letting the user quickly build models that are interoperable, independent of the other parts of the system (Mellor et al. 2004). In this author's understanding, this interoperability allows models to be more useful as a tool, rather than as a design artifact. Common concepts in MDE and MDA are explained in the following section.

C. CONCEPTS

Some general MBE concepts are summarized here. To provide the reader context; each is introduced and briefly described. These concepts are fundamental and independent of the modeling approach. The research in the following chapters explores the extent to which the approaches in scope implement these concepts.

1. Frameworks

A framework guides development through modeling or simulation by providing a structure for concepts and views (Balci 1988). In other words, a framework can provide a foundation for modeling and simulation scenarios that supports commonality and reuse, streamlining and tailoring tasks specific to the stakeholder. Typical frameworks for verification have formalisms for modeling and properties for verification and an algorithm for checking the system against a specification (Valmari 1998). Krogstie (2003) argues that a focus on frameworks and language quality is necessary to advance areas of modeling improvement. Modeling should be constructive, sharing relationships with linguistic theory, making models understandable and relatable to the stakeholders of a particular project utilizing modeling. Frameworks can define the use of schemas, an early example being the set of diagrams, which are used to draw electrical schematics (Ogren 2000). While frameworks pre-date computing, electronic schematics can now be edited and simulated by software tools.

Many popular architecture frameworks for MBE have evolved, each catering to different stakeholders based on their community's needs (Urbaczewski and Mrdalj 2006). Popular methods include DoDAF, Zachman, Federal Enterprise Architecture Framework (FEAF), Treasury Enterprise Architecture Framework (TEAF) and The Open Group Architectural Framework (TOGAF). While each framework specifies the use of models and viewpoints, there are many differences, such as scope and users, and whether time and motivating factors are inputs to the execution of the model (Urbaczewski and Mrdalj 2006).

2. Behavior Modeling and Emergent Behavior

Behavioral modeling is broadly defined and multidisciplinary. Among a few example categories, its application can be seen in systems and software engineering, biology, and astrophysics. For example, it can represent human behavior and interaction in the field of cognitive science (Penaloza et al. 2012). Astrophysicists use behavior models to represent the interaction of planetary orbits (Barnes and Greenberg 2007). One of many examples related to technological design can be seen in the use of a behavioral model as a method of performing an advanced Failure Modes and Effects Analysis (FMEA), which determines the potential failure mechanisms and their impact on the overall system (Eubanks, Kmenta, and Ishii, 1997). "[Behavioral models] are what the system *does* (how it behaves) as opposed to what the system *is* (which are models of form)" (Maier 2009, 232). A more specific definition is the measure of interaction (events) between components (Moshirpour et al. 2013).

According to Gore et al. (2007, 113), "Emergence can represent a valid behavior arising from seemingly unrelated phenomena, or it can reflect an error in a model or its implementation." Broadly, emergent behavior is only evident at a system level and not directly apparent as the resultant interaction of constituent entities (the individual parts of a system) (Checkland 1993). This is significant, as system level, behaviors can be missed in design, resulting in undesirable behavior. System level emergent behavior can contribute to cases where individual elements or sub-systems can lead to unexpected behaviors that cut across system boundaries, due to the coupling of structure and

behaviors (Grogan 2013). While there does not seem to be a consensus in the research reviewed as to how emergent behavior can arise, it is characterized as unexpected until explained and seen at a system level. Johnson (2006) describes some researchers as viewing emergence as simply unexpected behavior at one extreme, while others argue it is simply properties of a system that cannot be shown by functional decomposition.

3. Abstraction

According to Buede (2009), a model can be defined as an abstraction of reality, usually associated with the filtering of unnecessary detail for a given viewpoint. More simply put, "a model is an abstraction" (Krogstie 2012, 89). Abstraction is important for human understanding because of our finite level of attention, becoming indispensable when dealing with complex systems. MBE relies on abstraction as a method of reducing design and analysis complexity, as well as increasing comprehension. A model can be a tool for creating and exploiting abstraction (Kent 2002). In other words, abstraction can tailor a viewpoint to the user's needs. For example, an Internet browser does not show a user all of the various code used to process a web page, only content. Furthermore, many web pages are tailored between desktop viewing and smartphone viewing, the latter often in simpler format to accommodate a smaller screen and mobile nature. "Models of software requirements, structure and behavior at different levels of abstraction help all stakeholders deciding how this goal should be accomplished and maintained" (Mens and Van Gorp 2005, 126). Dale and Walker (1996) emphasize that most programming languages utilize abstract data types, which when used to define procedures (through semantics), separate the properties of the data type from its implementation details. An abstract model, using abstract data types to describe the underlying model, can be used to produce product specifications (Dale and Walker 1996).

When dealing with verification of a system, it can be more judicious to deal with a higher level of abstraction than with the details of the states in which one is interested (Valmari 1998). In other words, assertions or claims at a detailed level can be abstracted to a smaller level of claims on their properties. This allows fewer assertions to be checked while providing validation of more granular claims.

4. Separation of Concerns

Pressman (2015) defines a concern as a feature or behavior as specified by a model. Separation of concerns is a conceptual approach to dealing with complexity; problems can be "separated" (or modularized) and dealt with individually, reducing the perceived complexity (Pressman 2015). "This separation allows for the locality of different kinds of information in the programs, making them easier to write, understand, reuse and modify" (Hürsch and Lopes 1995, 1). The concept relates to the grouping (increased cohesion) of similar functions or traits of behavior and interaction. Combined with abstraction, separation of concerns is another powerful method of reducing complexity providing for different ways to organize and group considerations about a given model. For example, an electrical engineer on a project is not necessarily concerned with the structural properties of a product, whereas the opposite may be true for a mechanical engineer. Separating electrical from mechanical product specifications would result in simpler, easier to understand design from either individual's perspective.

5. Stepwise Refinement

Wirth (1971) details the importance of decomposing tasks into subtasks, refining steps into progressively smaller design decisions. Stepwise Refinement refers to the top-down process of elaboration; starting with a high-level function or requirement and gradually creates hierarchy through lower levels with more detail and less abstraction (Pressman 2015). Broadly defined, it deals with a progression from general and qualitative to specific and Quantitative (Maier 2009). By this definition, an example such as the systems engineering (SE) process of functional decomposition is a process of refinement.

6. Formal Methods and the Small Scope Hypothesis

Formal methods were born out of the software community and deal with developing systems in such a way that indicates functional and non-functional compliance with a given specification (Maier 2009). In the case of theorem proving, this can be a guarantee (Valmari 1998).

Methods for modeling the possible states a system can take are problematic; they can quickly overwhelm computing resources for even simple problems. This challenge is known as the state explosion problem (Valmari 1998). According to Valmari, there are two approaches to checking the correctness of a concurrent system against its specification: state space methods and theorem proving. Valmari describes state space methods as an automated method of determining the structure of all possible states reachable within a system. Theorem proving, in contrast, is the process of using mathematical formula(s) to show a measure of correctness (Valmari 1998).

The common problem that arises with either method is that even a small program's state space can be exponential. However, Jackson argues that certain approaches to model checking (searching for instances that violate a given property) can reveal errors in relatively few instances of a model. "Even a small scope defines a huge space, and thus often suffices to find subtle bugs" (Jackson 2012, 14). However, Jackson argues that this requires precise, unambiguous use of a formalized set of abstractions. Dolby et al. (2007) maintain that there is a compromise between static analysis and comprehensive testing when a model checker is focused on a specific property, called systematic under-approximation. Although this may limit the comprehensiveness of testing, it can reduce the complexity of the testing. A related paper, discussing the small scope hypothesis as applied to test set programs shows a high number of errors can be found through a small number of examples (Oetsch et al. 2012).

D. SUMMARY

General concepts related to MBE were summarized and defined. These serve to provide a baseline for understanding the common themes involved in using modeling for system design. The concepts, particularly abstraction and frameworks, are a broad means of dealing with increasing complexity in system development. The application of these concepts in various behavior modeling approaches is discussed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. BEHAVIOR MODELING APPROACHES

A. INTRODUCTION

This chapter serves to summarize the specific methodologies and claims of the primary sources surveyed in this research, representing the general state of multidisciplinary approaches to behavior modeling and simulation. Following the introduction of common MBSE concepts in the preceding chapter, two surveys related to formal methods (one specific to self-adaptive systems) adoption are summarized in this chapter. Based on these surveys and from this author's survey of sources pertaining to the concepts in the prior chapter, several modeling approaches are highlighted. These approaches are categorized into domain-specific and general approaches. A summary comparison of approach to the concepts discussed in Chapter II is provided after each is introduced. Additional supporting source material is used throughout this chapter to support analysis and conclusions.

B. FORMAL METHODS BACKGROUND

Use of formal methods is an approach to enable the creation of reliable systems despite the increased complexity. From the aspect of creating and verifying system specifications, formal specification makes use of a language with mathematically defined semantics, or syntax (Clarke 1996). As noted in Chapter II, the use of formal methods is noteworthy as a mechanism for distinguishing valid from invalid behavior, which can be useful in avoiding undesirable behavior. This section highlights two existing surveys found in the literature on where and how formal methods are used today.

1. Formal Methods Survey

Formal methods can be used to specify software and hardware requirements in any phase of the life cycle, utilizing abstraction where systems are complex. A comprehensive survey of the various applications of formal methods, focusing on the early application in product life cycles, claims to highlight the state of the art in their industrial application (Woodcock, et al. 2009). In this survey, the authors utilized a

questionnaire to gather data on 62 industrial projects known to use formal methods. The results were broken out by application domain, as seen in Figure 3.

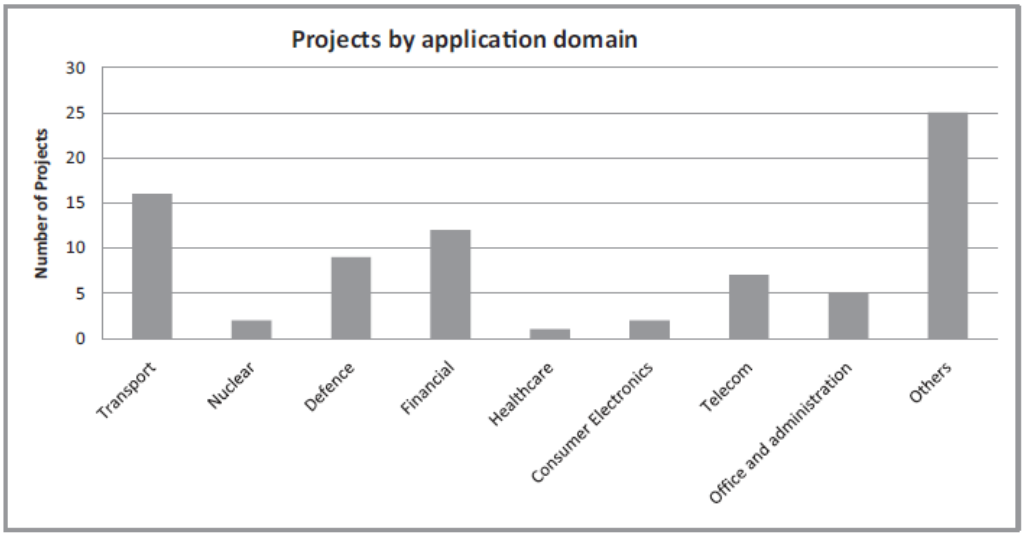


Figure 3. Projects Using Formal Methods by Domain
Source: Woodcock et al. (2009)

The results of the survey showed that for the majority of those surveyed (75%), the perceived effects on time, cost and quality were positive, leading to the desire to continue using formal methods on future projects. Of note, the authors argue that while the use and acceptance of formal methods appear to be rising, widespread adoption has not yet been seen outside of the development of critical systems. Some key takeaways the authors note is that formalism may not need to be applied equally to all components or stages of a product in development, and that more robust, automated tools are needed for wider acceptance.

2. Survey of Self-Adaptive Systems and Formal Methods

A second survey was reviewed, specific to the adoption of formal methods in self-adaptive systems (Weyns et al. 2012). The authors' goal was to identify the approaches, trends, tools and applications of formal methods used for self-adaptive systems. Self-adaptation is defined as the ability of a system to change its behavior as a result of a change in its perception of itself and the surrounding environment (De Lemos et al. 2013). Source material was limited to those that dealt with formal methods and separation

of concerns within the context self-adaptive systems; 1,027 of 6,353 studies were analyzed. Figure 4 and Figure 5 show the authors' findings concerning the trends of research broken out by venue and specification language.

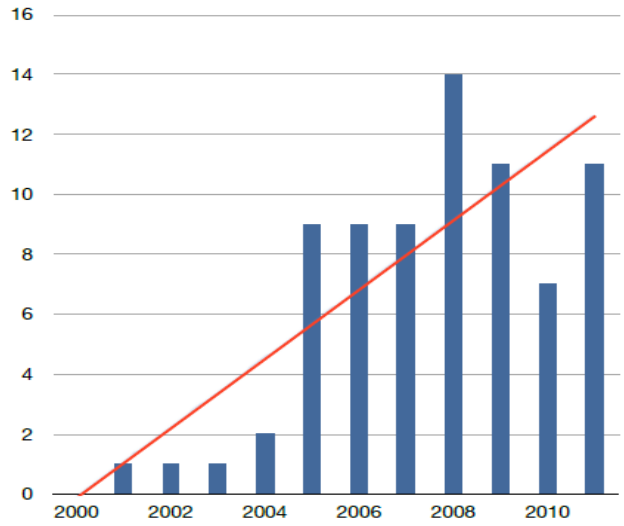


Figure 4. Formal Methods in Self-Adaptive Systems. Source: (Weyns et al.) 2012

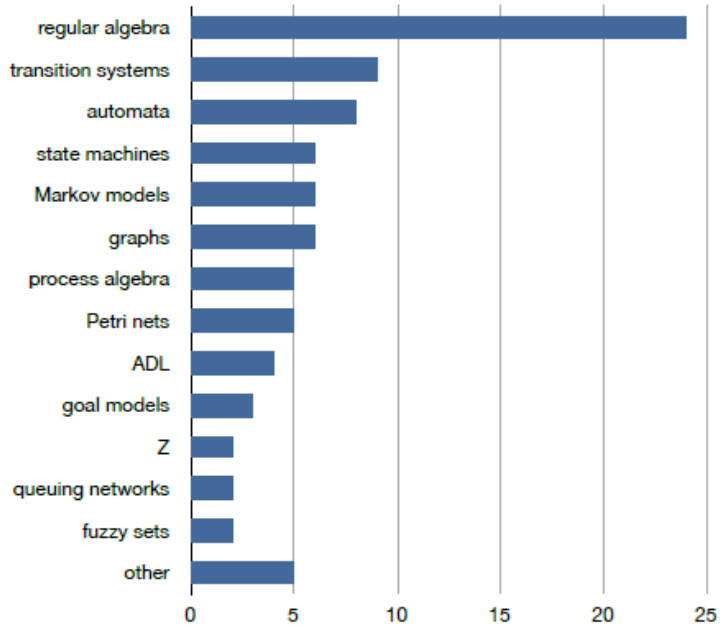


Figure 5. Specification Approaches Source: (Weyns et al.) 2012

Of note in the survey, the use of algebraic methods for formalization is the most common, and that there was no trend in approach over time. Additionally, 40% of the sources employed the use of tools, of which one-third was applied for model checking. The authors observe that the consideration of formal methods is lacking for the concerns of security and scalability and in the domains of telecommunication and scientific (climatic, bioinformatics) research. Figure 6 shows the distribution of verification properties that were the focus. Notable results were that the majority of researchers focused on formal methods for reasoning (as opposed to modeling or proving). Weyns et al. (2012) also note that "…only 23 studies employ formal methods to actually provide evidence for the self-adaptive concerns of interest." Weyns et al. remark that of the latter, one-third applied formal methods at run time, two-thirds applied offline, with only one study showing an application from design to runtime. Of interest, no sources surveyed covered the concerns of maintainability or portability (Weyns et al. 2012).

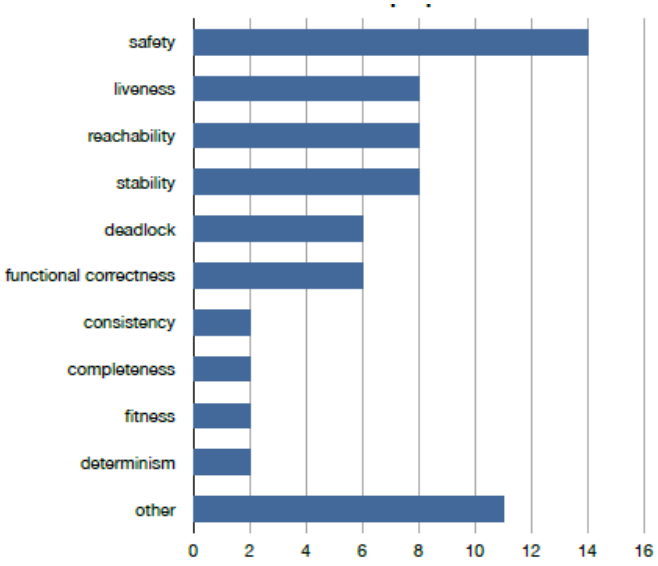


Figure 6. Formal Verification Properties. Source: Weyns et al. (2012)

Focusing on the studies concerned with modeling and model checking, the authors mapped the state space of self-adaptive behaviors to properties of interest, shown in Figure 7 (Weyns et al. 2012). The transitions between behavior states are of interest,

showing the properties involved and referenced sources on those properties. The authors offer this model as a reference point for future research and that tools for automated model checking, particularly at runtime, provide an underutilized opportunity for maturation. Noted is that few researchers provided publicly available models and results, which is offered as an indication of lack of integration in research. They conclude that many of the sources in the survey introduce modeling language constructs, but that the majority assume or ignore mathematical soundness, implying a lack of concern for formalism (Weyns et al. 2012).

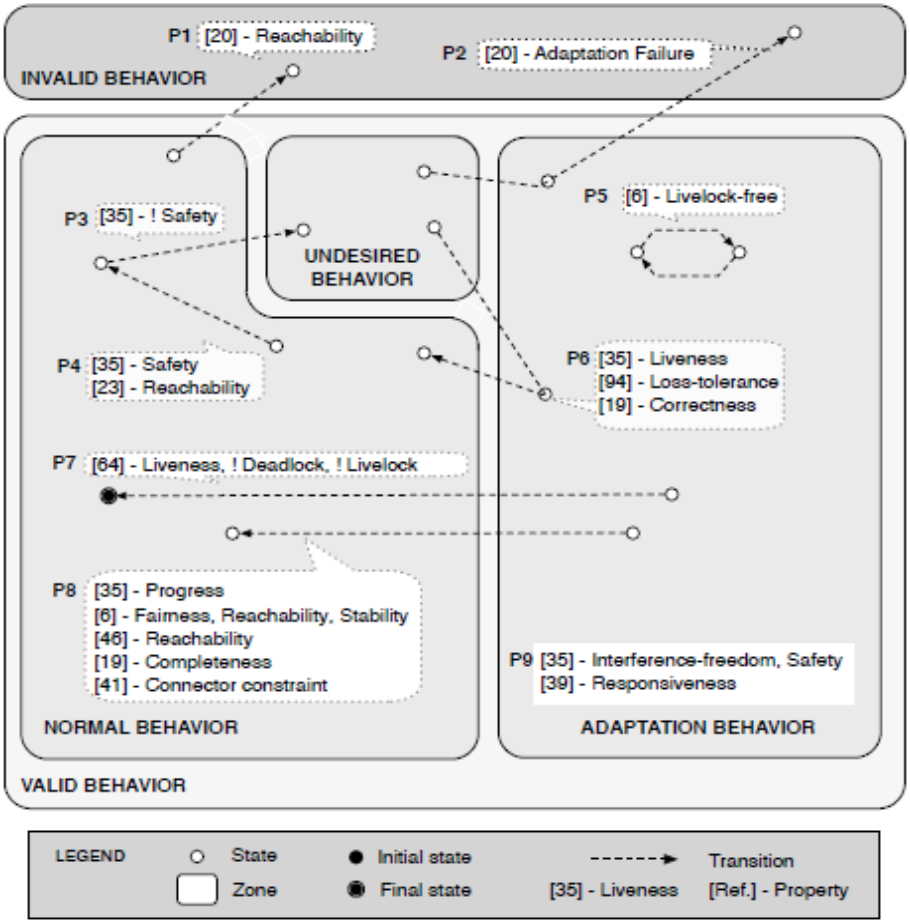


Figure 7. State Space of Behaviors to Properties, Numbered References. Source: Weyns et al. (2012)

C. DOMAIN-SPECIFIC APPROACHES

Moving on from the general state of the use of formal methods in modeling, this section covers two examples of domain-specific behavior modeling approaches in the literature. The automotive domain was chosen as an active area of research into the behavioral modeling of complex systems.

1. Automotive Requirements Modeling

In the automotive domain, a novel domain-specific approach for formal behavioral modeling is highlighted. Challenges are recognized in modeling and design approaches owing to increasing complexity of embedded software and electronics in automotive development, compounded by original equipment manufacturer (OEM) and supplier development taking place in isolation. One proposed method to address interoperability and timing issues, utilizing a form of Architecture Analysis and Design Language (AADL), uses an expressive timing relationship language and an expression of component-level requirements inclusive of validation (Yu et al. 2015). Yu et al. argue that safety and integration of continual new functions (in particular autonomous driving) will require design validation at the earliest possible phase to keep cost from being prohibitive. UML, SysML, Modelica, SCADE and MATLAB/Simulink are languages and tools that are commonly used for high-level modeling along with other heterogeneous models, the diversity of which the authors argue present a challenge to model integration. Integration solutions, such as AUTOSTAR (automotive) and System Architecture Virtual Integration (SAVI) are mentioned, but dismissed as problematic, suggesting integration frameworks instead. Virtual Integration is proposed as an approach to overcome the timing relation, component execution, composability and architectural constraints: a virtual model framework encompassing functions, architecture, viewpoints and optimization. Encompassing this approach is a "dual design" methodology called Inside-out and Outside-in; the first being concerned with decomposition into supplier contracts and the second focused on accomplishing an integration of subsystems that satisfies all contracts (Yu, et al. 2015).

The primary characteristics of the approach support are

- a system-level design model that encompasses both architecture and behavior
- an intermediate common formal model, which serves to facilitate semantics interoperability between dissimilar models through translation
- formal analysis, verification and formal timing specifications
- ual-design methodology, which decomposes a contract into sub-contracts and integration of sub-systems
- contract-based design, correct by construction and system optimization (Yu et al. 2015)

Figure 8 shows the authors' proposed integration framework. Key to this approach is contract-based design, which the authors describe as bringing further rigor into current automotive methods and practices. Two problems are suggested. The first problem identified is that models (specifically those that are translated into executable code) tend to be insensitive to differences in hardware characteristics, such as latencies and processor speeds; resulting in a departure from the platform characterization and the design.

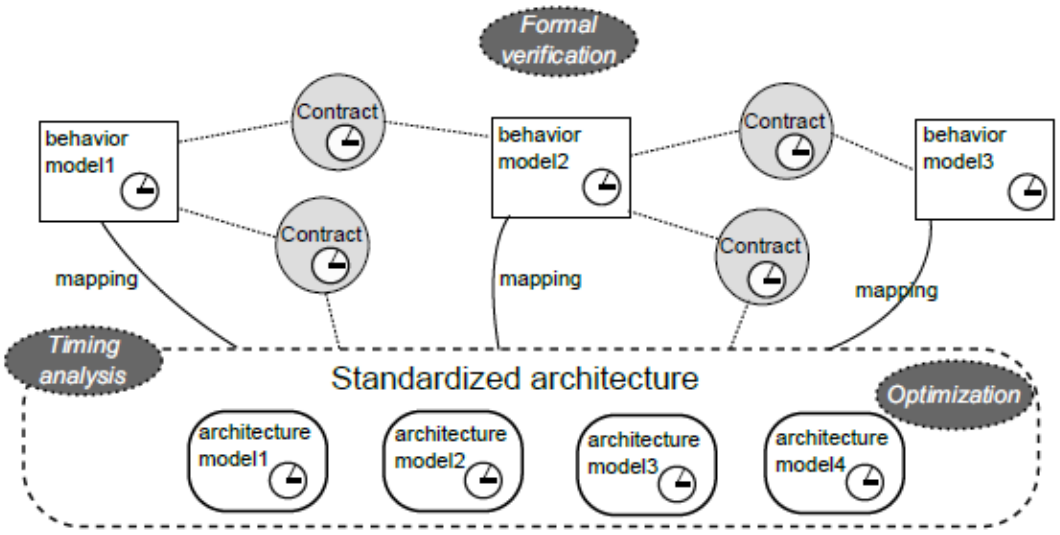


Figure 8. Proposed Automotive Architecture Framework.
Source: Yu et al. (2015)

The second problem identified is that while sub-systems may meet their individual requirements, this does not necessarily lead to successful integration, due to the unidentified subsystem to subsystem incompatibilities (including those that may arise in the future due to inflexible interfaces). The dual design approach mentioned above aims to solve these issues. Inside-out is an algorithmic decomposition process of system level properties to provide sub-system contracts (vice natural language) to suppliers, with the goal of exactly meeting the supplier contract. Outside-in relates to the supplier's viewpoint, ensuring that the sub-system contracts are designed to meet optimally the platform characteristics. The authors conclude that further adoption of their framework approach, in conjunction with AADL and the associated forthcoming synchronous timing annex, will help to overcome the integration challenges discussed in the first part of the paper.

The framework proposed by Yu et al. (2015) is noteworthy as it proposes a modeling methodology to improve issues noted in automotive development, such as system verification and requirement specification at the lowest and highest levels of hierarchy. Multiple modeling languages are proposed depending on the viewpoint, such as Simulink as the primary behavior modeling approach, specifically focused on modeling timing and synchronicity as hardware and software events. The software architecture is defined by AADL. Hardware and software timing constraints are represented by logical and algebraic abstractions. Separation of concerns is not explicitly mentioned; however, it is noted that different tools and models are better suited for the separate areas of optimization, behavior, and contract specifications. A forthcoming timing annex to AADL will be used to apply formalism to both the architecture and the behavior models (Yu et al. 2015).

2. Automotive Control Modeling

Sengtacken, DeLaurentis, and Akbarzadeh-T (2007) present a novel approach to utilizing behavior models in a hypothetical automotive control system. They present a potential future of wirelessly interconnected autonomous vehicles (swarm) and offer a notional method of balancing the level of human versus autonomous control. In this case

study, an agent-based modeling (ABM) framework is employed as a control system to determine the level of autonomy between the driver and the car, with consideration given to other vehicles, the sensors employed and the environment (Sengstacken, DeLaurentis and Akbarzadeh-T 2007). ABM is discussed in more detail in a forthcoming section.

Shown in Figure 9, two agents are assigned to model the driver, while two agents model the control system. The driver agents are goal-oriented, supporting abstract deduction, with a deliberately slow reaction time. The control agents are modeled as fuzzy logic rule sets to represent vehicle's autonomy, and the relationship between the two offers the ability to explore shared autonomy approaches (Sengstacken, DeLaurentis and Akbarzadeh-T 2007). In the study, fuzzy logic algorithms are used for simulating control of path tracking, obstacle avoidance, and information feedback. A swarm of six vehicles was then simulated, varying the degree of human versus machine control over the same functions; showing a method of minimizing hazards (crashes) while maintaining an element of human control (Sengstacken, DeLaurentis and Akbarzadeh-T 2007).

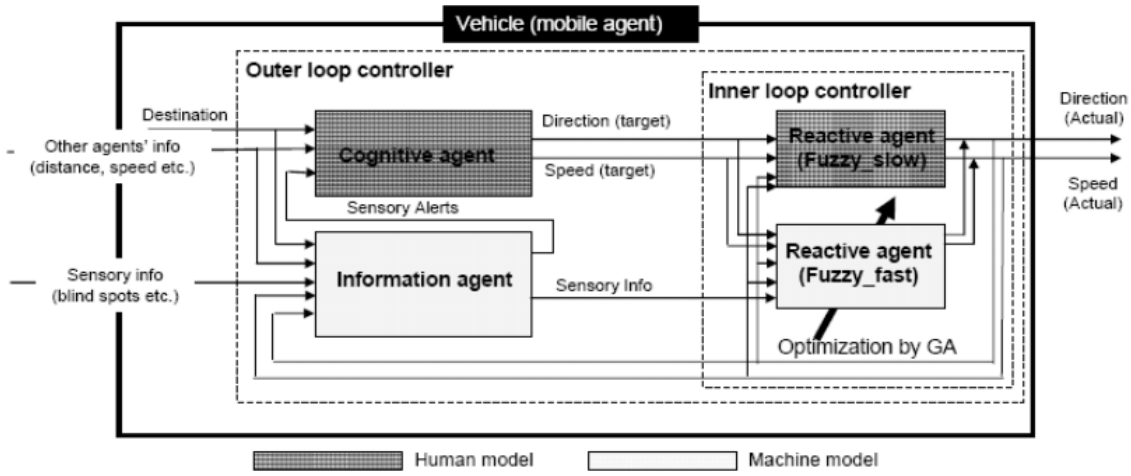


Figure 9. Automotive Control System Utilizing Agents
Source: Sengstacken, DeLaurentis, and Akbarzadeh-T (2007)

The framework described by Sengstacken, DeLaurentis, and Akbarzadeh-T (2007) proposes a method of quickly exploring automotive control architecture changes with respect to achieving the desired balance of human and vehicle control. Behavior is

modeled by the use of fuzzy logic-based agents, abstracted to essential sensor information and control functions. The concepts, stepwise refinement, and formal methods are not expressed or apparent in the paper (Sengstacken, DeLaurentis, and Akbarzadeh-T 2007). Abstraction is not explicitly discussed, but may be inherent, as the level of abstraction is tailorable with the use of ABM (Borshchev and Filippov 2004). Additionally, separation of concerns is not mentioned explicitly, though the agents are separated by role (human, machine) as well as further decomposed into internal/external stimuli (Sengstacken, DeLaurentis and Akbarzadeh-T 2007).

D. GENERAL APPROACHES

In this section, the following modeling constructs are explored to provide coverage of a range of approaches to behavior modeling.

1. Systems Modeling Language

The Object Management Group (OMG) adopted the first version (1.0) of UML in 1997 as a standardized modeling language to aid in software development (Kobryn 1999). Systems Modeling Language is a customized version of UML adopted in 2006, tailored to the needs of engineers (specifically related to requirements linkage) (Hause 2006). The SysML specification contains four primary means (viewpoints) of expressing behavior, borrowed from UML – activity, sequence, use case, and the state machine diagrams (Fowler 2004). The SysML diagram hierarchy, showing differences from UML, can be seen in Figure 10. Of these, activity and state machine diagrams (circled in the figure) are best suited for representing behavior, as the use case and sequence diagrams are focused more on the specifics of interaction. An example activity diagram can be seen in Figure 11.

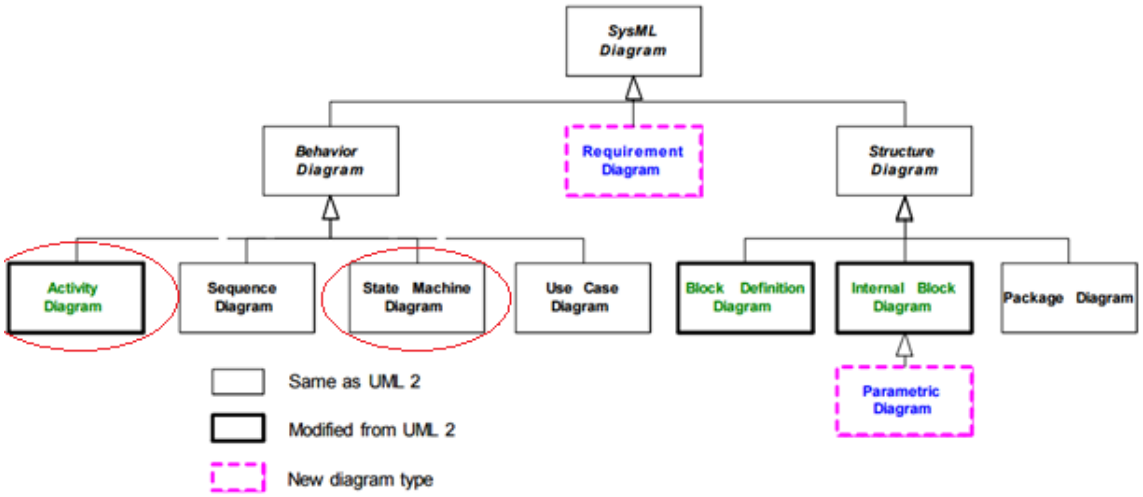


Figure 10. SysML Diagram Hierarchy. Source OMG (2015)

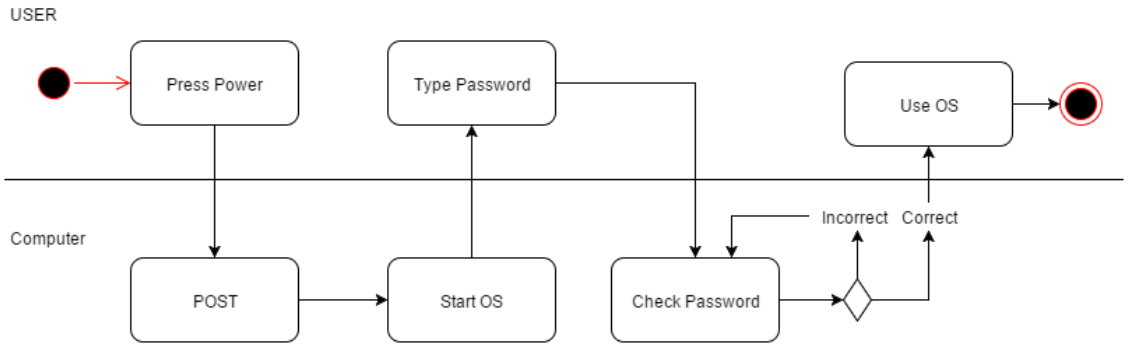


Figure 11. Activity Diagram

The framework defined by the SysML specification is a general-purpose design and analysis language, supporting behavioral, structural and requirements modeling (OMG 2015). Models can be designed at the desired level of abstraction, and can be refined to lower levels of detail. Separation of concerns can be achieved using multiple viewpoints, but results in at least one diagram per actor, with inputs and outputs between each. Stepwise refinement is not an inherent feature of SysML though Miyazawa and Cavalcanti propose a method of utilizing refinement as an extension of SysML (2014). The use of formal methods is not inherently supported by the current specification. However, Graves and Bijan give an example of integrating SysML with formal logic-based semantics to minimize inconsistencies and support assertion checking (2011).

2. System Dynamics Models

System Dynamics (SD) arose from the concepts of control and feedback in order to model some of the first computer simulations in the late 1950s (Forrester 1993). Figure 12 shows the steps in the process, beginning with formulating the problem to be solved (generally correcting for undesired behavior in a system), and formulating equations to describe the system. Feedback is central to the system dynamic approach: each step is iterative and recursive, drawing on and improving the prior steps (Forrester 1993).

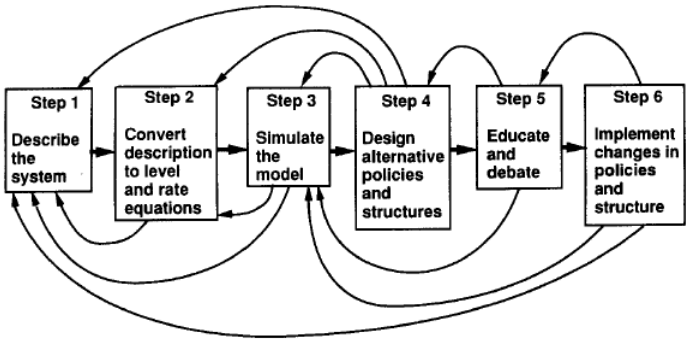


Figure 12. System Dynamics Process. Source: Forrester (1993)

System Dynamics deals with complex behavior in which the system can be non-linear, is constantly changing, actors are coupled, is self-organizing, is adaptive, and past events govern future events (Sterman 2001). Figure 13 provides an example SD model. In this example, Sterman develops equations that govern positive and negative feedback for each actor, which can be simulated to show the behavior of an adoption system. Over time, different actors dominate the system flow, which can be used by a decision maker to change their business practices (Sterman 2001).

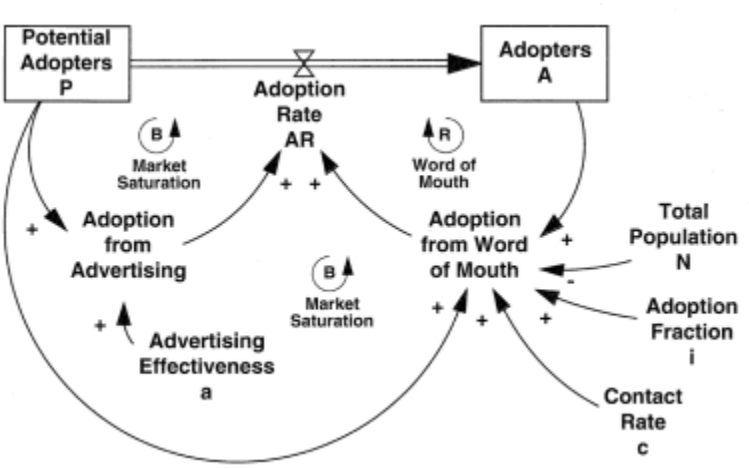


Figure 13. Example SD Model. Source: Sterman (2001)

Systems Dynamics is a general modeling approach. While there are shared core concepts of conceptualization and notations, differing frameworks and approaches exist (Martinez-Moyano and Richardson 2013). System behavior is represented by flows, levels and causal loops (Forrester 1993). System Dynamics models are abstract by nature, as they are limited to closed loop systems and the number of causal factors. The ability to separate concerns is difficult when causal factors are differing but inter-relate (Forrester 1993). The use of stepwise refinement is not explicitly mentioned in the literature. While the publications reviewed did not directly mention the use of formal methods with respect to SD, the models are built upon mathematical equations. The resulting parameters can be compared against real-world observables (Forrester 1993).

3. Monterey Phoenix

Monterey Phoenix is a novel approach to systems and software architecting, and process modeling. Monterey Phoenix began as an approach to specification modeling of software system architecture with behavior models (Auguston 2009). Auguston's premise is that a major concern of software architecture design is capturing the behavior of the system. This is proposed through utilizing behavior models, represented as a set of event traces to model system requirements with formalized event grammar. Following the initial paper, MP has been expanded from a software architecture approach to include

system architecture and business process and workflow modeling (Auguston 2014). The grammar is combined with constraints to produce schemas, which are an executable representation of a system's architecture (Auguston 2014).

While there are many existing tools and languages for software and system architecture modeling and design, one of the primary characteristics of MP is attempting to solve the common "single flowchart" problem (Auguston et al. 2015). Instead of attempting to capture all behavior or activity in one place, MP produces an output for each possibility (an exhaustive set of the event traces). Event traces have two basic relations: precedence and inclusion (Auguston 2015). System behavior is defined by the set of event traces that satisfy event grammars and constraints, collectively the schema (Auguston et al. 2015). A schema is defined by MP source code, with formal, structured syntax.

According to Auguston (2015), an MP schema contains one or more root events, where trace derivation starts. Event grammar rules define relationships and constraints when applying composition operations on the traces assembled from root events. "A grammar rule specifies structure for a particular event type (in terms of `IN` and `PRECEDES` relations)" (Auguston 2009, 1032). Sequencing of events can be controlled by the order of these relations. Figure 14 shows a simple event trace example. If `IN` is shown as a solid arrow and `PRECEDES` is shown as a dashed arrow, the rule A: B C shows that the parent event A is an ordered set of event B preceding the occurrence of event C. Operators can change the relationships between events, some examples of which can be seen in Table 1. There are many more MP expressions, and composition operations used to tailor execution of a model, such as Boolean operators, assigning event probability, variables and assertion checking (Auguston et al. 2015). However, behavior models can produce complex results solely with the syntax introduced.

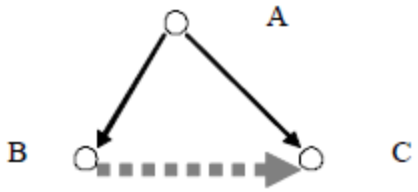


Figure 14. Event Trace A:B C (note that MP Analyzer notation uses dashed for inclusion, and solid for precedence) Source: Auguston (2009)

Table 1. Example MP Operators. Adapted from NPS (2015)

Natural Language Description of Pattern	**Pattern Expressed as MP Event Grammar Rule**
Unordered set of zero or more events B	A: {* B *};
Unordered set of one or more events B	A: {+ B +};
Unordered set of events B and C (B and C may happen concurrently)	A: {B, C};
Ordered sequence of zero or more events B	A: (* B *);
Ordered sequence of one or more events B	A: (+ B +);
Ordered sequence of events (B followed by C)	A: B C;
Optional event (B or no event at all)	A: [B];
Alternative events (B or C)	A: (B | C);

Monterey Phoenix models are executable by way of an event trace generator (Auguston 2015). At the beginning of model execution, for each root event described, all valid traces within the scope *N* (provided by the user) are derived, with dependencies determining event order. Following the small-scope hypothesis, *N* can be relatively small (single digits) and produce the majority of possible states. Traces are produced based on the order of appearance of root events, while considering composition operations, such as SHARE ALL or COORDINATE. Constraints such as ENSURE allow more efficient execution by pruning state space search. The set of traces produced within the scope and constraints provided can be analyzed for examples of unexpected behavior. These counterexamples can then be removed by modifying the model's code (e.g., through the ordering of events or modifying constraints), starting the process of process of validation early in the life cycle (Auguston 2015). Chapter IV gives an example MP model.

Auguston (2015) describes the MP framework as facilitating an architecture that serves to facilitate bridging requirements and implementation. Models allow for simulation of system behavior and demonstration of potential emergent properties.

Models at the system level show the earliest form of design, facilitating analysis of alternatives. Abstraction is central to MP, in that architecture descriptions should ignore implementation details. Separation of concerns is achieved by separating the interaction description from the behavior. Furthermore, MP supports execution of the architecture specification through stepwise refinement (Auguston 2015).

4. Agent-Based Modeling

Agent-based modeling and simulation (ABM) is a broadly defined and emerging method of modeling complex systems using "agents" with defined behavior(s) and the relationship(s) between one another and the environment (Macal 2010). In general, agents interact (influence one another), are unique (facilitating heterogeneous populations), autonomous (independent) and have time-varying states. Optional agent characteristics can include adaptivity (an agent may or may not "learn" from its behavior) and goal-orientedness (seeking outcomes). An example application of agents in automotive control was seen earlier in Section C.2.

Agent-based modeling approaches, definitions, and applied disciplines span a wide array of designs and applications, and there are not yet broadly accepted or standardized definitions for ABM (Borshchev and Filippov 2004). Further highlighting the breadth of ABM, agent attributes and the rules applied to them can range from extremely simple, static to complex and dynamic (Borshchev and Filippov 2004). Common features of ABM approaches include a decentralized system representation, and localized agent interaction and information sharing. The sum of this interaction is the topology (connectedness), changing over time as agents change their behavior or their neighbors. The topology can be represented in a number of different fashions, such as linked networks, in two or three-dimensional space or as unrelated to spatial dimensions (random interaction). The environment can serve as a simple reference, or it can impose its own information or constraints on the agents.

As an ABM model is decentralized, system behavior emerges from agent interactions, rather than being explicitly defined by the model. This leads to ABM being called a "bottom-up" modeling approach (Borshchev and Filippov 2004). Assuming

agent behavior attributes are known, modeling can be done without a solid understanding of system interdependencies and behavior. "ABM replicates independent agents in order to study their interdependencies" (Baldwin 2015, 365).

In addition to the example shown in Section C.2, another example application of ABM can be seen in the study of mathematical biology. There is recognition in the usefulness in ABM to simulate complex, dynamic and heterogeneous biological systems (such as tumors). However, Hinkelmann et al. argue that tools for mathematical analysis of discrete models in biological applications are limited, mostly restricted to qualitative versus quantitative analysis (2011). The authors argue that the Overview, Design Concepts and Details (ODD) protocol is a potential solution to this problem.

ABM is a broadly defined and applied methodology for modeling the dynamics of systems. Some aspects are similar to MP – both are primarily concerned with the behavior of complex systems. In particular, MP and ABM share separation of concerns of agent behaviors and agent interactions, in theory capturing latent behavior that may not easily be found with other methods. However, ABM currently lacks the level of standardization and representation needed for formal verification. ABM is inherently "bottom-up" and inherently difficult to verify (Pullum 2012).

As discussed, ABM approaches vary by domain and application, and there are multiple frameworks and specific approaches. System level behavior is seen as a result of agent interaction, showing emergent behavior in simulation results. According to Baldwin, ABM is used at all levels of abstraction, from granular elements, such as pedestrians, to complex entities, such as companies. From the literature reviewed, the relationship of ABM to the concepts of separation of concerns, stepwise refinement and formal methods is unclear at this time, warranting further research. In particular, none of the literature reviewed explicitly mentions the small scope hypothesis in conjunction with ABM.

5. Petri Nets

One specific area of research into formal models of self-adaptive systems focuses on capturing emergent requirements, specifically at runtime using Petri nets. There is a

well-established body of research on Petri net modeling, dating to the 1960s (Krogstie 2012). Petri nets are stepwise, iterative and concurrent state transition systems. The basic components of a Petri net, a form of a directed bipartite graph, are *states* (also called places) and *transitions*, with flow denoted by *arcs*. One or more *tokens* traverse a Petri net by *firing*, indicating the occurrence of transitions (Petri and Reisig 2008). Petri nets are well established as a behavior modeling approach, facilitating graphical representation based on formal semantics (Jensen 2013). According to Jackson, a static model only describes a system's states, while a dynamic model also describes the system's transitions (2012). By this definition, Petri nets are dynamic models.

One issue with traditional Petri nets is the problem of modeling systems with a large number of states, causing the model itself to become complex. A method of dealing with this comes in the form of colored Petri nets (CPN). A CPN is one type of high-level Petri net that allows tokens and transitions to be assigned colors (and variable information), which permits differences in routing behavior, as well as more compact models (Jensen 2013).

Some characteristics are shared with MP: abstraction (via encapsulation), mathematical formalism, and graphical notation. They also provide hierarchical representation with abstraction in many different domains with a variety of tools (Girault and Valk 2013). Petri nets are also being used in the study of business workflow modeling, an example of which can be seen in Self-Adaptive Recovery Nets (SARN) (Rachid and Benatallah 2004). Separation of concerns can be accomplished with Petri nets (Abdelzad and Aliee 2010). However, whereas MP explicitly separates processes for each actor (interleaving them later to produce scenarios), a secondary method or process must be applied using Petri nets to segregate aspects or behaviors from one another. MP and Petri nets will be further compared by contrasting a simple model in Chapter IV.

E. SUMMARY

This chapter summarized selected behavior modeling approaches, each of which was then related to the concepts in Chapter II. The surveys presented some general trends in the application of formal methods broken out by domain and approach. Examples of

domain-specific frameworks were summarized, introducing novel concepts such as virtual integration and multi-scale modeling.

General behavioral modeling approaches were introduced. Core characteristics of Monterey Phoenix were summarized. It is shown that MP shares common characteristics of MBSE. In summary, the aim of MP is in leveraging these traits to deal with the challenges of unpredictable behavior resulting from complexity, while overcoming some of the recognized shortfalls of current approaches. Some semantics were introduced to impart an understanding of MP code and usage. Agent-Based Modeling is introduced. Finally, Petri nets are introduced, discussing the basic concepts of one of the older and more popular behavioral modeling approaches.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PETRI NETS COMPARED AND CONTRASTED WITH MP

A. INTRODUCTION

To support the summary of behavior modeling approaches introduced in Chapter III, simple experimentation was undertaken. This chapter compares MP with Petri nets by way of a basic model, to compare and contrast the two approaches. Petri nets were chosen, in part, because it is shown as an area of current research in the formal methods survey presented in Chapter III, as well as sharing some characteristics and goals with MP. The selected MP model is shown first, followed by an approximate Petri model, methodology, and comparison of results.

B. EXAMPLE MODEL—MP COMPARED WITH PETRI NETS

1. MP Example

To minimize errors in interpretation introduced by model translation and varying toolsets, a rudimentary behavior model for communication between two nodes is utilized for a baseline comparison. The scenario is commonplace and well understood. The schema for the model and a prototype MP simulation environment, MP Analyzer, can be found and executed online (NPS 2015). An alternative online tool, Eagle6, is also accessible (Rivera Consulting 2015). However, at the time of this writing, it does not implement `COORDINATE` statement. The MP schema for the model is as follows:

```
SCHEMA Communication
ROOT A_getDataFrom_B:     (*   A_requestDataFrom_B
                          A_waitDataFrom_B
                          (    A_failReceivingDataFrom_B    |
                          A_receiveDataFrom_B )       *);
ROOT B_answerRequestDataFrom_A:    (*                    (
                          B_receiveDataFrom_A
                          (    (B_processDataFrom_A
                               B_sendDataTo_A ) |
                               B_failProcessingDataFrom_A)
                          |
                               B_failReceivingDataFrom_A )
                          *);
COORDINATE $x: A_requestDataFrom_B FROM A_getDataFrom_B,
               $y:  (    B_receiveDataFrom_A|
               B_failReceivingDataFrom_A
               FROM B_answerRequestDataFrom_A
               DO ADD $x PRECEDES $y; OD;
COORDINATE     $x:     (     B_failReceivingDataFrom_A     |
     B_failProcessingDataFrom_A )
                    FROM B_answerRequestDataFrom_A,
               $y: A_failReceivingDataFrom_B
                    FROM A_getDataFrom_B
          DO ADD $x PRECEDES $y; OD;
COORDINATE     $x: B_sendDataTo_A
                    FROM B_answerRequestDataFrom_A,
               $y: A_receiveDataFrom_B
                    FROM A_getDataFrom_B
          DO ADD $x PRECEDES $y; OD;
```

This example model has two root events, node A requesting data from node B (“`A_getDataFrom_B`”) and B answering the request for data to A (“`B_answerRequestDataFrom_A`”). The “`A: (* B *)`” notation denotes that A contains an ordered sequence of zero or more B events. Child Events of each root follow and end with a semicolon. The “`COORDINATE`” syntax is a composition operation that coordinates the behaviors of the root events in terms of message passing (Auguston, Behavior Models for Software Architecture 2014). Alternatively, a “`SHARE ALL`” statement (not used in this example) could be used: if there is event type of one or more of the events, and there is an event trace that satisfies the schema, both root nodes share those event types (Auguston 2009). For example, if there is a satisfactory event trace

solution that contains "A_requestDataFrom_B," it could be shared between both root events "A_activity" and "B_Activity."

2. MP Example Results

Every MP model can be run through a number of iterations, or its event scope (set at two in this case). Following the small scope hypothesis, the event scope is set at two and the model executed. The results are a set of 13 possible event traces (which satisfy the schema), listed in Appendix A. An example output is shown in Figure 15. Each box represents an event (green boxes are root events, blue are the resulting composite events). The dashed arrows represent inclusion (IN) (e.g., "A_requestDataFrom_B" is a composite event included in the root event "A_getDataFrom_B"). The solid arrows represent precedence (PRECEDES) (e.g., "A_requestDataFrom_B" is an event preceding "A_waitDataFrom_B"). This example shows a failure, followed by a successful transmission of data. While there 13 event traces, it should be noted that all but three involved another iteration of the model from the starting event "A_requestDataFrom_B." In every instance, the trace will arrive at either "A_failReceivingDataFrom_B" or "A_receiveDataFrom_B." This explanation discounts the event trace that only contains the root events (as explained earlier, this is due to the * operator), because, for the purpose of this comparison, some activity is assumed to take place. Alternatively, events with zero instances would be removed if the (* … *) operator was replaced with the (+ … +) operator, as described in Chapter III.

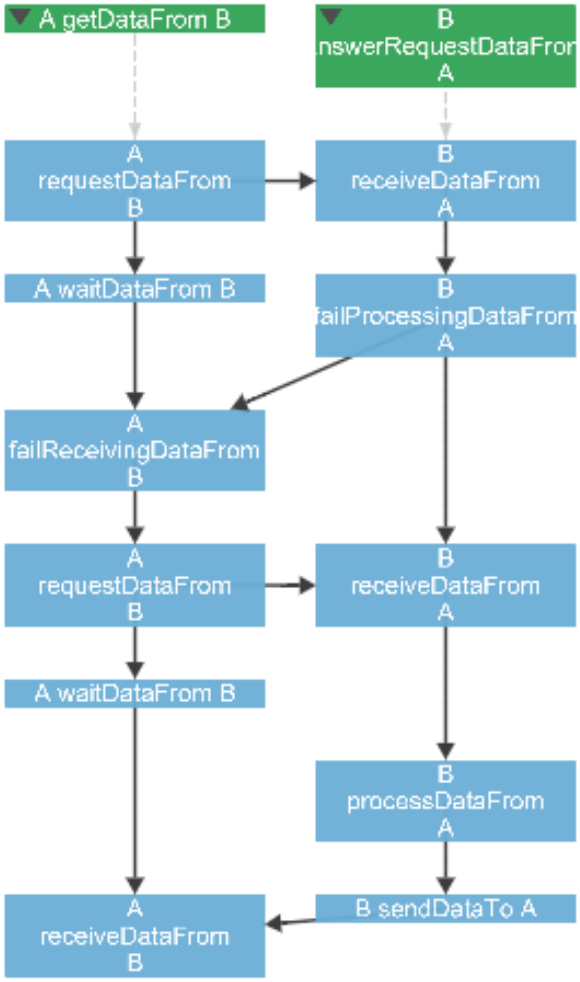


Figure 15. Example Event Trace Output

3. Petri Net Example

Of the numerous Petri net tools available, Platform Independent Petri net Editor 2 (PIPE2) was chosen for being platform independent (using JAVA), currently supported, open-source, and supporting analysis modules such as reachability, state space and timing (Bloom et al. 2007). The behavior model for communication between two nodes was manually translated from MP into a Petri net with equivalent states (Figure 16). The MP model event names were preserved for clarity. To maintain simplicity and equivalency with the MP model, arcs (which can be modified to allow more than one token at a time) are set at the default of one. Likewise, transitions (which can have timing properties) are

set at a delay of zero. The two MP root events are not included, as they are always assumed to occur.

As MP is only concerned with events, which may be considered abstractions of states, it should be noted that all states (places – Petri net circles) are actually describing the transition(s) preceding that state. For example, "A_receive_dataFrom_B" properly worded in Petri convention would be "A_receivedDataFrom_B." It should be noted that the event labels could have been assigned to transitions; however, this would have necessitated adding additional states, as tokens can only be assigned to states. Supporting this choice, Raschke (2009) shows some success in formal translation of UML activity diagrams into token-based state machines.

Figure 17 shows a closed loop variation, which allows continuous running of the model. The closed loop variation involved the addition of three transitions and nine arcs. One of these arcs is an inhibitor arc (going into transition T11), which enables a transition to fire only when a token is not present at the preceding state (however, T11 will not fire without the other arc also receiving a token from state "A_failReceivingDataFrom_B"). This allows the model to synchronize after an iteration, because in some cases, multiple tokens can be present. After an iteration, only a single token will be present at the beginning state "A_requestDataFrom_B." A single token was placed in the "A_requestDataFrom_B" state, and 500 firings were performed on each model.

The observed difference between the open and closed loop models is that the open loop shows tangible states – that is, possible end states. For example, the open loop model has three states, S5, S6 and S8 (corresponding to "A_failReceivingDataFrom_B," "A_waitDataFrom_B," and "A_receiveDataFrom_B," respectively) that a token will terminate (no more possible firings). However, S5 and S6 are essentially a shared end-state, since they will always occur together. This is discussed further in the next section.

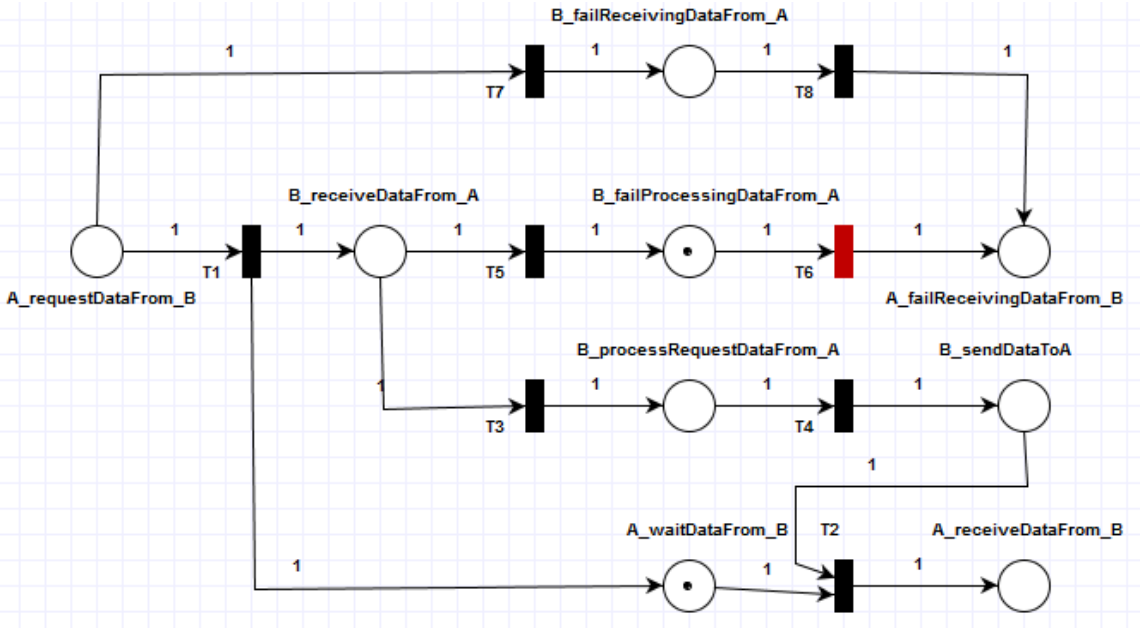


Figure 16. Petri Net Data Transfer Example (Open Loop)

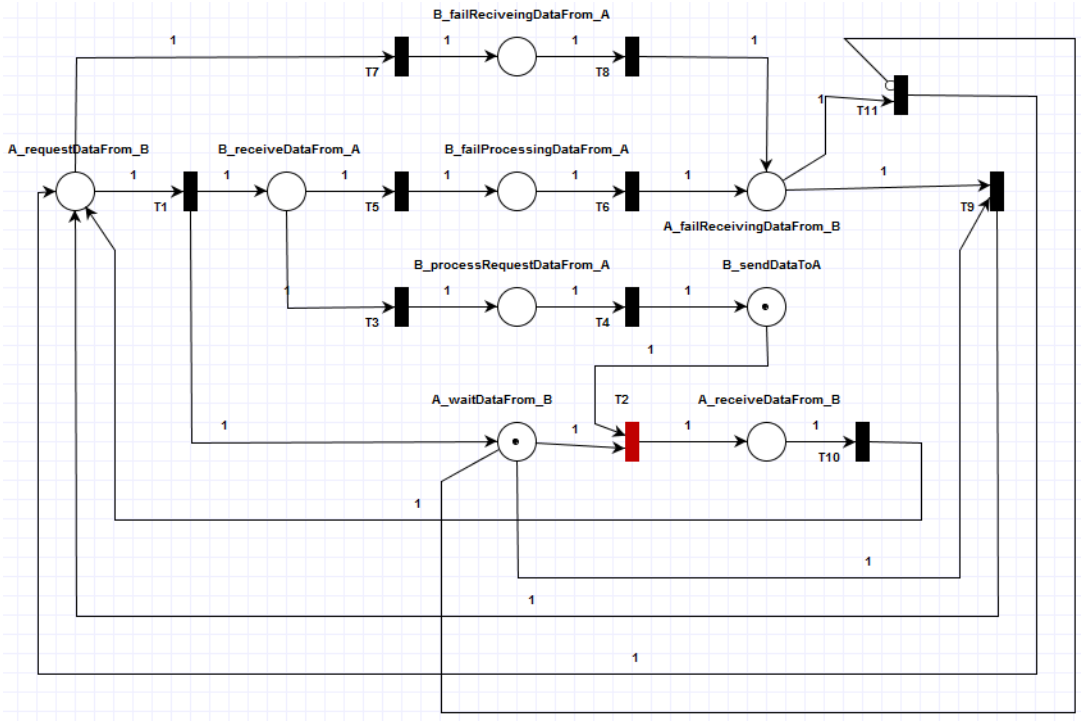


Figure 17. Petri Net Data Transfer Example (Closed Loop)

4. Results and Comparison with MP Example

From a behavioral perspective, reachability is defined as the entirety of possible states (i.e., those states that are fireable, or where a token is able to progress), supported by mathematical formalism (Lambert 1992). A reachability report was run on both the open loop and closed loop models, seen in Figure 18 and Figure 19 respectively. In comparison with the MP model results, all possible states are shared. This was verified by comparing the MP model outputs to the sequence of firings seen for each run of the Petri net models. PIPE2 does not support generating a graphical output of individual iterations; however, the model was observed to transition through each of the MP example event traces (found in Appendix A). The one exception is the case of only the root events, with no observed behavior. This is because the (* *) operator applied to the root events specifies iteration of its contents zero or more times. In contrast, the Petri net will always fire if the transition is enabled by a token (set by default at the beginning state "`A_requestDataFrom_B`."

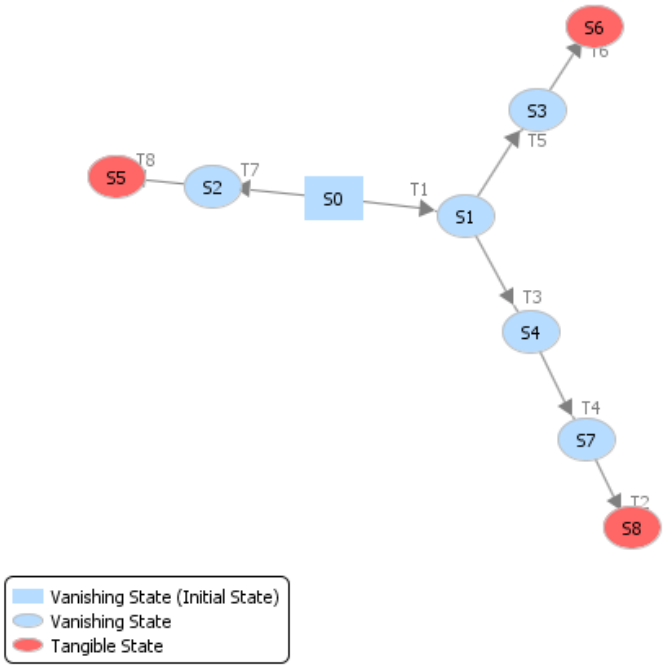


Figure 18. Reachability Output (Open Loop)

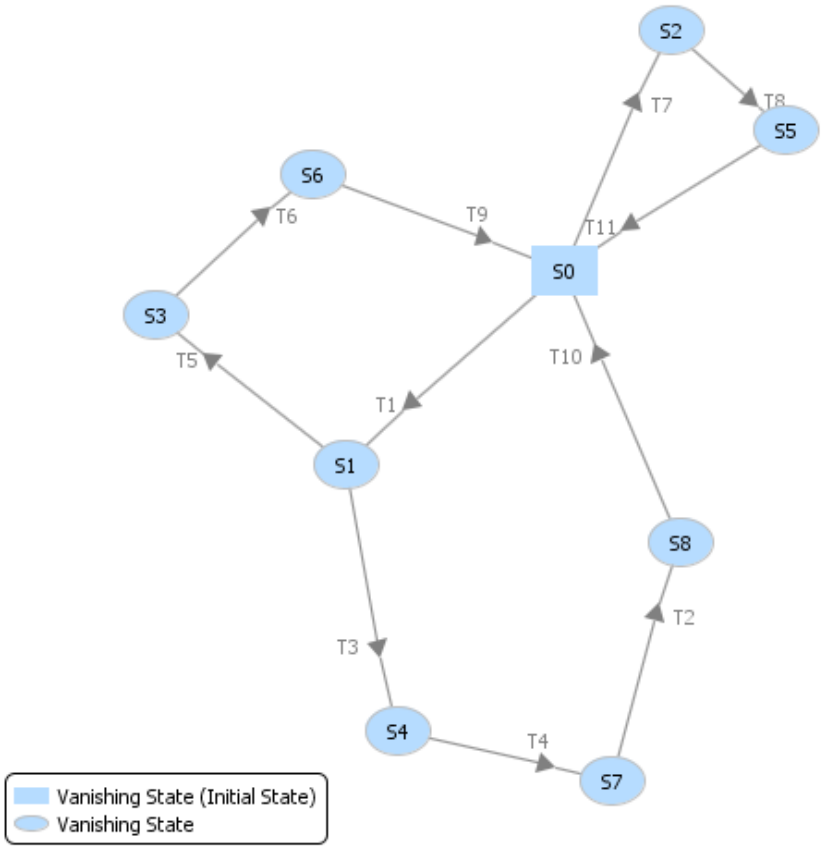


Figure 19. Reachability Output (Closed Loop)

In comparison to the MP results, an observed difference is that "A_waitDataFrom_B" is shown as preceding "A_failReceivingDataFrom_B." The open loop Petri net model, in this case, will terminate with a token in each state, which means that A will continue waiting for data from B indefinitely. In the MP model, a transition is shown to "A_failReceivingDataFrom_B" from both "A_waitDataFrom_B" and "B_failReceivingDataFrom_A." In other words, both the MP and the Petri net models share two primary end states, but in the case of the open loop Petri net model, an additional token is "stuck" at "A_waitDataFrom_B" in one of the two end states. A more complex model could potentially account for this, matching the MP results more closely. There are three possible end states for a token because an end state is possible with tokens in both S5 and S6. Otherwise, the observed transitions between the two Petri net models matched. In contrast, the closed loop model accounts for this state and is a better translation of the MP model. This results in a single

token beginning again at "`A_requestDataFrom_B`," aligning with the MP results as a new transition is started.

One difficulty encountered in translation of the MP model into a Petri net was the relative increase in complexity. Simple MP statements can result in a comparatively intricate Petri net representation. As an example (unrelated to the experimentation model), in root event D, starting from State A, if there is the possibility of a transition to B and then to C (or B again) or to C and then to B (or C again), or to both B and C simultaneously, all must be accounted for. An illustration of this as a Petri net can be seen in Figure 20 (tokens fired once or twice). In MP, this would simply be accomplished with the syntax `D: A (+ ( B | C ) +);` where D includes A, which precedes B or C. The Petri net model is relatively complex without the use of extensions, such as CPN. This is an example of the "single flowchart" paradigm, which increases the difficulty of accurately designing and maintaining a model, resulting from combining aspects of all or most of a system's behavior into one viewpoint (Auguston, et.al. 2015). The experimentation discussed in this chapter suggests that Petri nets (without extensions) suffer from this issue, as far more time was spent designing and debugging the Petri model relative to the MP model.

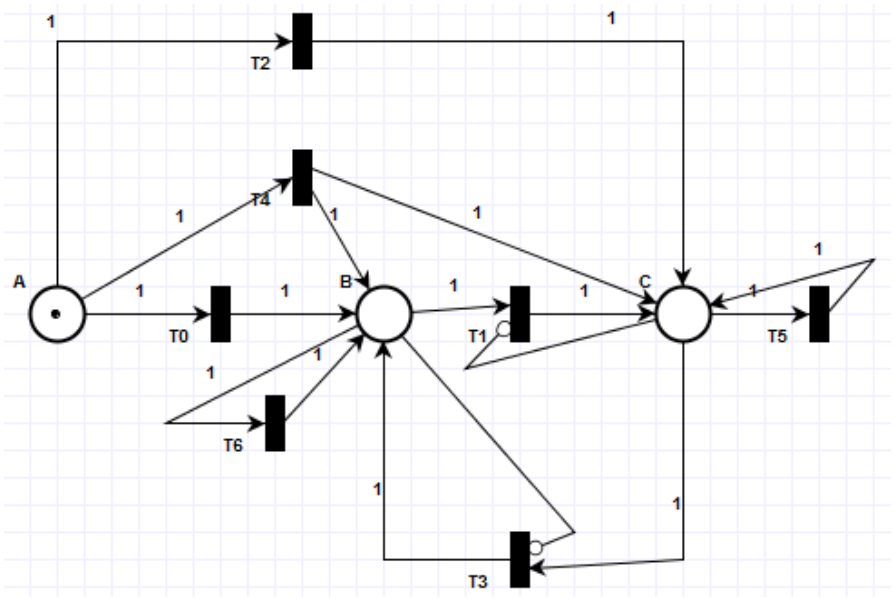


Figure 20. Petri Net Transition Example

Finally, a classification report was run for each variant, seen in Figure 21 and Figure 22. Of note, the open loop model is a free choice net (which means there is more than one potential path a token can take), while the closed loop model is not. In other words, the closed-loop net shows both concurrency and conflict between the transitions of tokens. A free choice net means that every transition from an arc to a place or a place to an arc is unique (Sgroi et al. 1999).

Comparing the observed transitions of the Petri net models and the MP event traces, all end states were shared. This implies that, at least for a simple example, both approaches can be made to show identical results of the behavior of a system. It should be noted that this does not necessarily indicate the approaches are equivalent in analysis capability. On the contrary, there was a qualitative difference in ease of use and time of experimentation. A more complex scenario may make it unwieldy to replicate in Petri nets, as inherent guidance to separate the concerns of each root event is not explicitly provided in Petri nets in comparison to MP. For this author, the MP model setup and run time was faster than with the Petri nets, even though the Petri net model provided a visual editor (versus MP source code). These differences could be attributable to the tools and not the frameworks. As the MP framework is refined and grows to support more features (e.g., timing constraints and measurements), future research should be undertaken to compare and contrast MP with other executable discrete modeling approaches. Specifically, a more complex model (with more possible end states, or more than two root events) should be compared to a Petri net or similar model. Additionally, the Petri net model in this research did not attempt to explicitly model `A` as a separate entity from `B`, whereas the MP model parses `A` and `B` independently from one another and interlaces the results at execution. PIPE2 has the capability of abstracting `A` from `B`, but it is not known if they could be executed independently but interlaced into one another, which future experimentation could show.

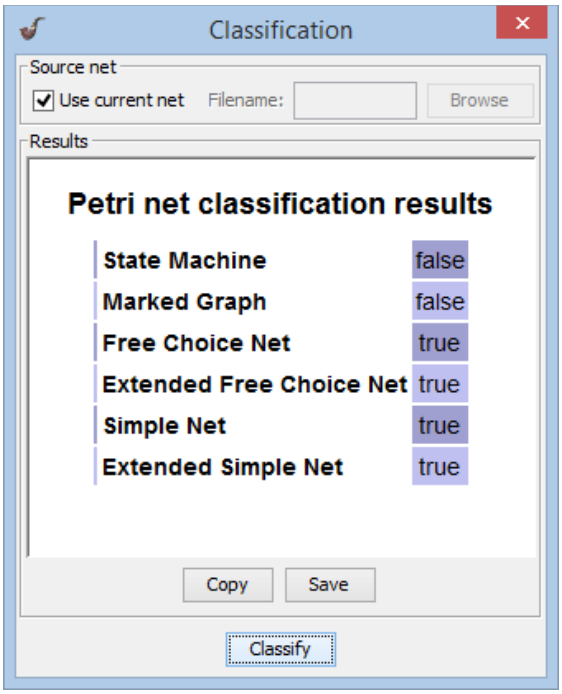


Figure 21. Classification Results Closed Loop Model

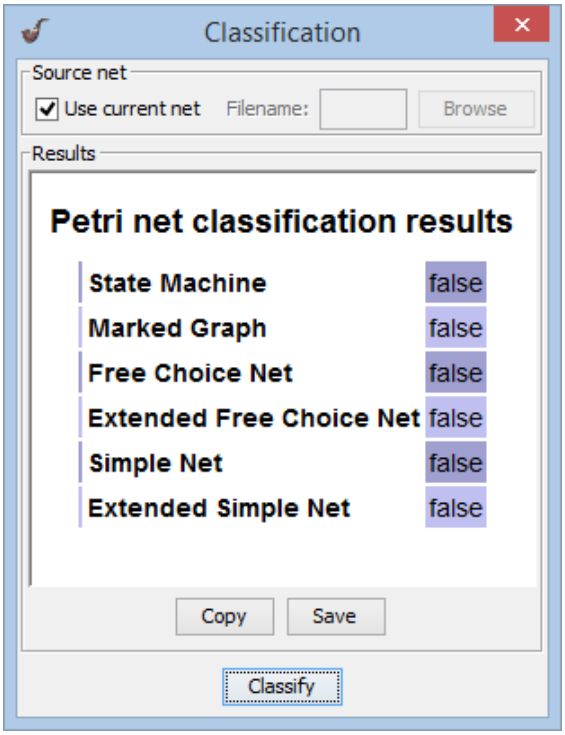


Figure 22. Classification Results Open Loop Model

C. SUMMARY

This chapter introduced a simple MP model describing the communication between two systems. Utilizing the online MP environment MP Analyzer, a schema for the model was executed and analyzed for possible events within a given scope. The MP model was translated into an approximate Petri net model with two variations and executed in the PIPE2 simulator. The primary difference between the two is that the MP model treats the two root events as independent of one another, modeling their interactions at execution. In contrast, the Petri net models treated the root events as a single dependent model. However, in this simple example, the resulting state spaces in the MP model and the closed loop Petri net model are shown to be logically equivalent. It is not implied that the approaches are equal in nature, only that they share characteristics that can be used to produce the same logical state space.

V. CONCLUSIONS AND FUTURE RESEARCH

A. INTRODUCTION

The intent of this research was to capture key modern MBSE approaches with respect to behavior modeling and in the context of MP characteristics. Throughout this thesis, many of the common concepts introduced in Chapter II are significant attributes of the approaches presented. The research question is revisited, followed by a summary of comparisons between concepts and reviewed approaches. The conclusions section presents notable insights from the comparison of approaches and the experimental comparison.

B. RESEARCH QUESTION

The primary research question is as follows: "How does the Monterey Phoenix behavior modeling approach compare with other approaches that claim to expose unintended, unneeded or undesired system interactions?" The research was accomplished through a review of the literature on the current state of behavioral modeling, contrasting with MP through a comparison of key concepts, and through experimentation with MP and Petri net models.

C. CONCLUSIONS

The following conclusions arise from review of the researched publications:

1. MP makes use of the concepts covered in Chapter II at least as much as any other method or framework reviewed.

2. Of the publications reviewed, no other formal, executable approach claims to exhaustively search for all possible scenarios (within a given scope) while also supporting event attributes, assertion checking, and different viewpoints.

3. The *Virtual Integration concept described by* Yu et al. may benefit from the use of MP, as the framework makes use of multiple modeling languages depending on viewpoint and is concerned with precise behavior modeling.

4. None of the approaches researched fully support all of the concepts (without the use of extensions or multiple specifications/approaches) reviewed in this thesis as MP does: frameworks, abstraction, behavior

modeling, abstraction, separation of concerns, stepwise refinement and formal methods/small scope hypothesis.

Table 2 shows a summary of researched approaches and the relationship of each to the general concepts described in Chapter II. Relations are drawn where supported by the research. Some relations are inconclusive based on this research, and as such are marked as "unclear" or "not explicit" with an explanation. These areas may be particularly supportive of future research.

Table 2. Select Approaches Related to Concepts

	Automotive Requirements	**Automotive Control**	**SysML Activity**	**System Dynamics**	**Monterey Phoenix**	**Agent-Based Modeling**	**Petri nets**
Frameworks	Approach is an integrated framework of methods. Architecture is defined by AADL	Control framework	Framework well defined as part of a common specification	General approach, multiple frameworks	Framework for verifiable behavioral modeling	Many differing frameworks	Well defined framework, extensions such as colored and timed Petri nets
Abstraction	Logical and algebraic abstractions for timing and synchronicity	Not explicit, may be inherent to the use of ABM	Inherent to approach	High level of abstraction at system level	Inherent to approach	Abstraction at the agent level	Limited to encapsulation
Behavior Modeling	Simulink for event / timing, forthcoming timing AADL annexes for constraints/ formalism	Uses agents to model behavior of a control system	Inherent to approach	Inherent to approach	Inherent to approach; primarily concerned with behavioral interactions	Agents used to model emergent behavior as seen at the system level	Discrete event based, can show potential state spaces
Separation of Concerns	Inherent to approach—framework ties together differing models, tools	Not explicit, though agents are separated by role and function	Not explicit	Unclear	Inherent to approach	Separation of agent behaviors from interactions	"Single flowchart" problem, requires secondary method(s)
Stepwise Refinement	Inherent to approach	Unclear	Not explicit, but method proposed for formalized refinement	Limited/ difficult	Inherent to approach	Unclear	Limited to encapsulation
Formal Methods and Small Scope Hypothesis	Proposed, not discussed in detail (forthcoming timing annex). Since multiple languages are utilized, MP could be applied to formalize behavioral aspects	Agents are mathematically described	Limited, extensions must be used. One such proposed extension cited.	Actors defined through differential equations	Inherent to approach; formal schema and syntax	Unclear. Some formal methods proposed, however inherently difficult to validate; small scope hypothesis not discussed in literature	Inherent to approach, mathematically defined

Expanding upon Conclusion 1, MP is the only approach researched that fully employs all of the concepts summarized in Chapter II. All of the general approaches are well reflected as being used for behavioral modeling across domains. However, each approach has notable gaps. One example is the use of formal methods in ABM. While well suited towards qualitative simulations with many unknowns, the lack of qualitative ABM methods can present difficulty in verification. In contrast to this, SD and Petri nets are well suited for quantitative problems, as models are described mathematically. However, SD does not appear to have practical methods for separation of concerns or stepwise refinement, and Petri nets are limited (encapsulation for stepwise refinement) or require extensions (e.g., colored Petri nets) to deal with separation of concerns effectively.

In further depth, experimentation with Petri nets and MP are shown to produce a logically equivalent state-space (reachability) for a simple communications model. However, the Petri model did not treat the root events (system A and B) as separate entities, as MP does prior to execution. This is notable, as the state transitions internal to each node are not distinguished from the state transitions between nodes. In this case, the states of A and B are combined into a single model that does not show a measure of coupling or cohesion between nodes. In contrast, MP separates component behavior from the interaction of components (applying separation of concerns). Most significantly, all possible state transitions and the order in which they can occur must be explicitly modeled in a Petri net. While MP must account for all possible states, the possible transitions and the order in which they can occur can be stated more abstractly.

A Virtual Integration approach was discussed in Chapter III (Yu et al. 2015). In common with MP, the authors are supportive of some of the same goals of abstraction, formal specification, and separation of concerns, with emphasis on timing. Attributes of *correct by construction* and contract extraction are distinctive to their approach. Of note, the authors recognize that the key characteristic of semantics interoperability may not be feasible due to the constraint of semantics preservation. The authors call for the use of behavior models that link supplier specifications to the system specification. These models are referred to generically, so MP may be of benefit to such a construct. A key

assumption is made in assuming suppliers are able to provide designs that optimally meet the functional and non-functional requirements in the contracts that they are provided. While this may be more achievable in the automotive industry, it is a questionable assumption the defense industry, given the proliferation and mandates applying to the use of Commercial Off-The-Shelf (COTS) (Oberndorf 1998). This off-the-shelf usage can impose additional complexity and incompatibilities and may restrict the flexibility such an approach would require (Alves 2003). However, MP may be suitable for use in the proposed virtual integration framework, as it is composable, testable and executable, and may be suited to the task of identifying subsystem-to-subsystem incompatibilities.

The comparison of approaches in Chapter IV shows that both approaches can produce identical state space possibilities, at least in the case of the simple communications model in this research. However, it does not conclusively show that MP and Petri nets are equivalent in studying the behavior of a system. One of the benefits of Petri-nets is the ability to pass a defined number of tokens at each decision point, but it requires explicitly defining at each state. Mentioned in Chapter IV, Petri net editors can support features such as control and measurement of timing, arc capacity. While these are not utilized in the example models, they are apparent candidate features for future MP environments. Additionally, Petri net tools allow graphical editing, while existing MP tools currently rely on building a schema. However, from experimentation, it appears Petri net models currently suffer from the "single flowchart" paradigm, increasing the effort relative to MP to design and maintain equivalent models (Auguston et.al. 2015).

D. LIMITATIONS OF RESEARCH

As discussed in Chapter I, this thesis is primarily focused on reviewing state-of-the-art approaches to exposing latent and undesired behavior. As such, limited experimentation was performed to supplement multi-disciplinary findings through literature review and surveys of current methods.

Sources and content are generally limited to recent research. Despite the multi-disciplinary and multi-domain track of this thesis, few sources dealt with concrete, comprehensive, real-world experimentation. One particular example modeling approach,

ABM, appears to have near-ubiquitous adoption across science and industry. Counterintuitively, ABM appears to have very little standardization across domains, particularly in verification and validation.

Although the results in Chapter IV show that a Petri net model and an equivalent MP model can produce identical state spaces, the models are simplistic, and further testing of models that are more complex should be done to verify the breadth of this conclusion. Additional comparisons between MP and Petri net with increased complexity, including the use of abstraction in a Petri net model would add more evidence to the relationship between MP and Petri nets.

ABM was only qualitatively explored in this thesis, explicit experimentation comparing MP to analogous MP models would further help to relate or distinguish the two approaches. As there is no currently published research relating MP to ABM, the relationship is inconclusive, warranting further research.

E. FUTURE RESEARCH

The SAE standard AADL, discussed in Section III, may offer further insight into formalized semantics (Yu et al. 2015). Additionally, a future synchronous behavior annex to AADL is mentioned, which may be of interest, though it is domain specific, conflicting with the desire for MP to be domain-independent.

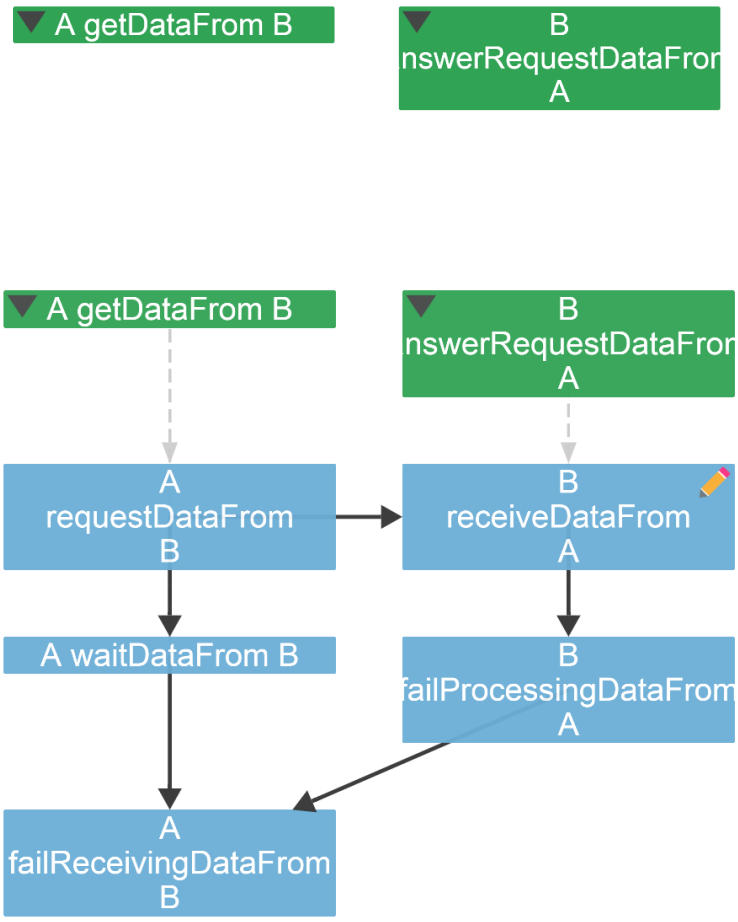
Chapter III provided a review of two comprehensive surveys of the state of research on formal methods, with one specifically focused on self-adaptive systems. The surveys highlight growing interest in formal methods applied to MBSE while presenting some issues seen in research and adoption. A key question Woodcock et al. ask is whether tailoring of when and where formal methods are applied in product development would aid in increased adoption of formal methods based modeling (Woodcock, et al. 2009). It follows that future research into the suitability of MP as applied specific development areas would help to build a stronger case for adoption. A recent study investigating the utility of MP for Business Process Modeling (BPM) is a good example (Auguston et al. 2015).

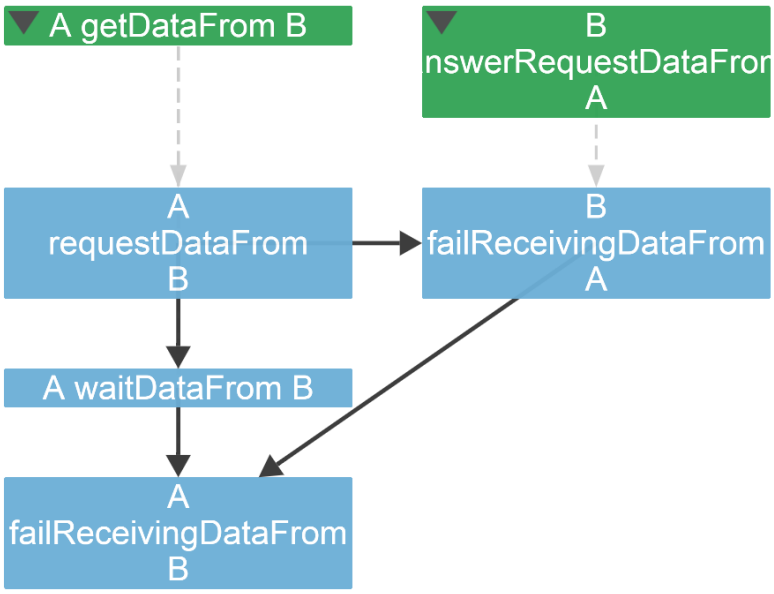
Of note, none of the sources focused specifically on modeling applications for the concern of maintainability or life-cycle cost. Further research on how MP could be applied to maintainability scenarios, perhaps building on existing reliability studies, would be of benefit to the defense industry. Model-based reasoning (MBR) is a knowledge-based method that has shown promise in significantly reducing the ambiguity (increasing the accuracy) of fault diagnosis and reporting (Berenji, Wang and Saxena 2005). Future research investigating the potential utility of MP in the study of fault propagation or as a complementary toolset for MBR is a good example, given ever-increasing Reliability and Maintainability (R&M) standards.

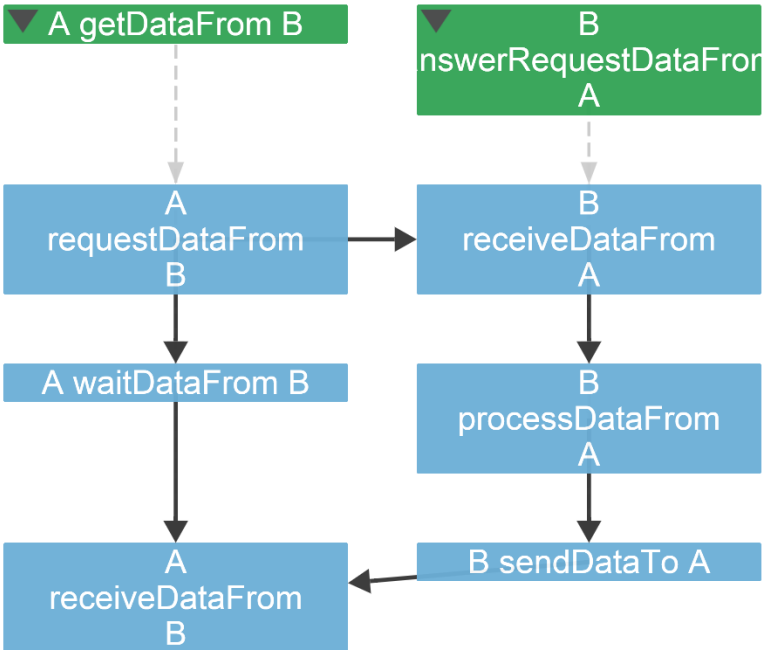
Further research into the design of MP tools is warranted. As shown with the ability to perform timing analysis with Petri nets, and any number of other modeling and simulation environments, capturing more than just the potential state space is useful. From a tool perspective, the potential to capture other behavioral traits, such as temporality should be explored. Additionally, a method of translating graphical objects into MP schema into would provide an additional method of creating MP models. Monterey Phoenix has an established baseline of syntax, but it is currently growing to accommodate additional use cases (for example, assertion checking). Furthermore, efforts to enhance MP Analyzer with additional views of event traces could aid in the analysis of results. One such example would be analogous to the reachability output graph of the Petri net PIPE2 tool as seen in this thesis – showing a summary of all possible states in one view.

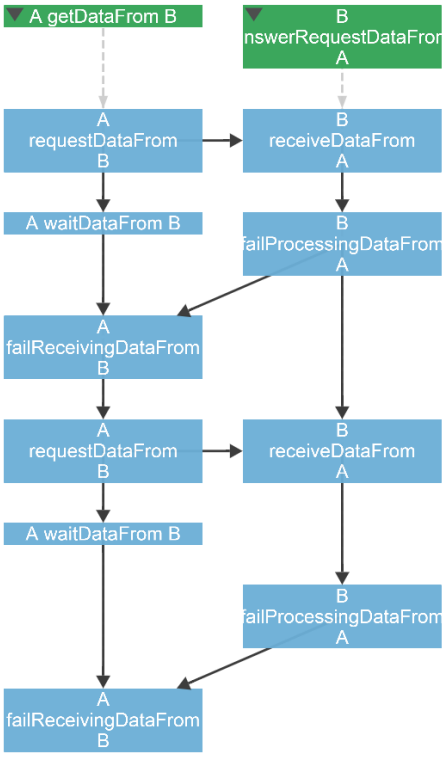
An up-to-date listing of MP information published research and presentations can be found at the following link: https://wiki.nps.edu/display/MP/Bibliography. An experimental web-based Graphical User Interface (GUI) for designing and running MP Analyzer can be found at http://firebird.nps.edu/. A practical venue of future effort would simply be for interested parties to contribute to the documentation supporting MP (particularly known use cases, standards, tools), which would aid most future research.

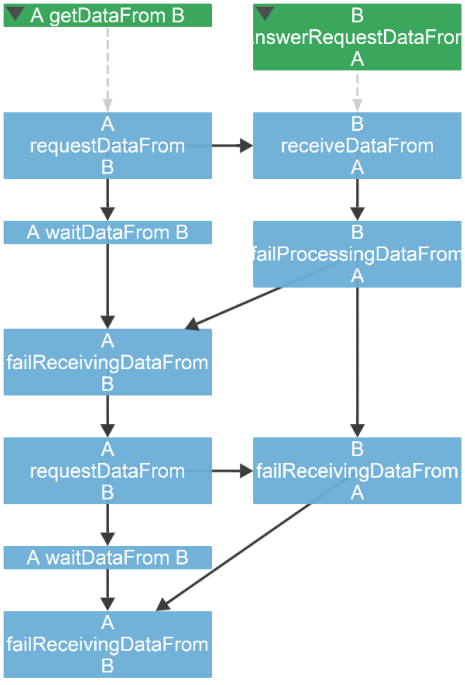
APPENDIX. MP MODEL RESULTS

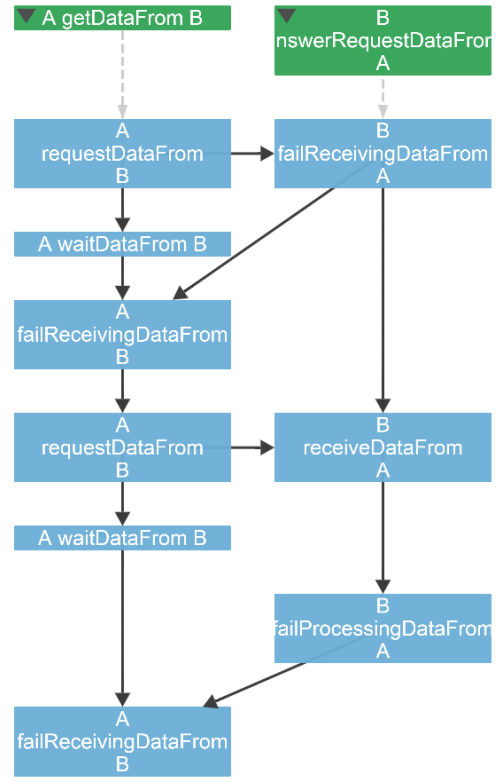


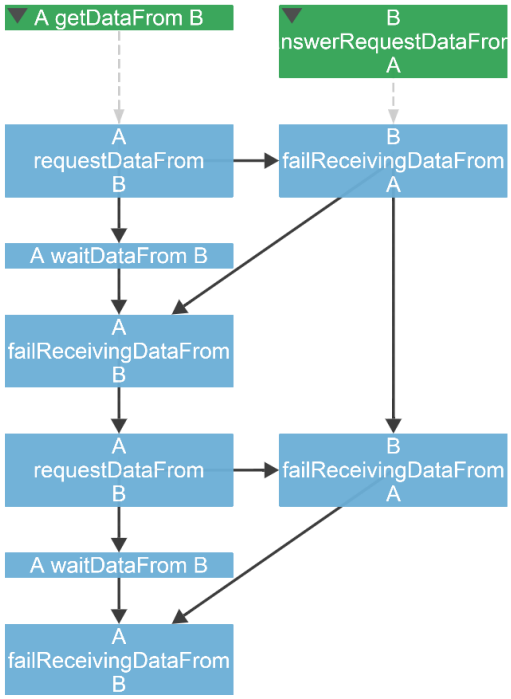

A getDataFrom B
B
nswerRequestDataFror
A
A
requestDataFrom
B
B
failReceivingDataFrom
A
A waitDataFrom B
A
failReceivingDataFrom
B

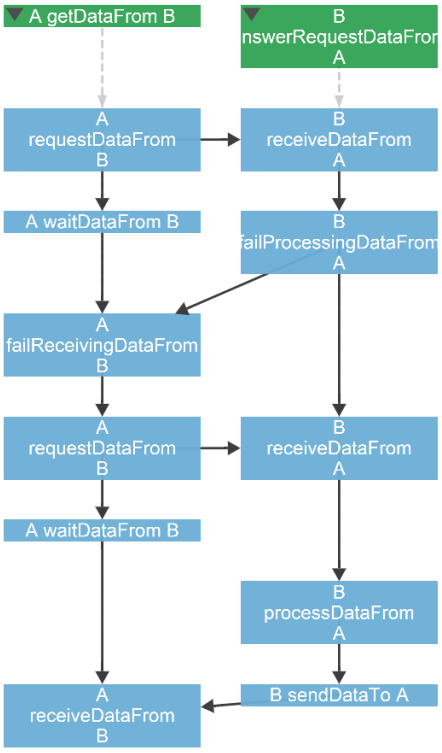

A getDataFrom B
B
nswerRequestDataFror
A
A
requestDataFrom
B
B
receiveDataFrom
A
A waitDataFrom B
B
processDataFrom
A
B sendDataTo A
A
receiveDataFrom
B

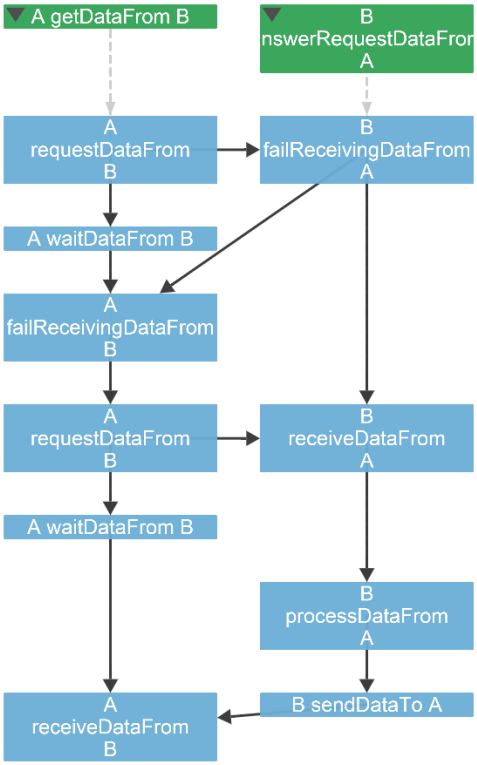

A getDataFrom B
B
nswerRequestDataFrom
A
A
requestDataFrom
B
B
receiveDataFrom
A
A waitDataFrom B
B
ailProcessingDataFrom
A
A
failReceivingDataFrom
B
A
requestDataFrom
B
B
receiveDataFrom
A
A waitDataFrom B
B
ailProcessingDataFrom
A
A
failReceivingDataFrom
B

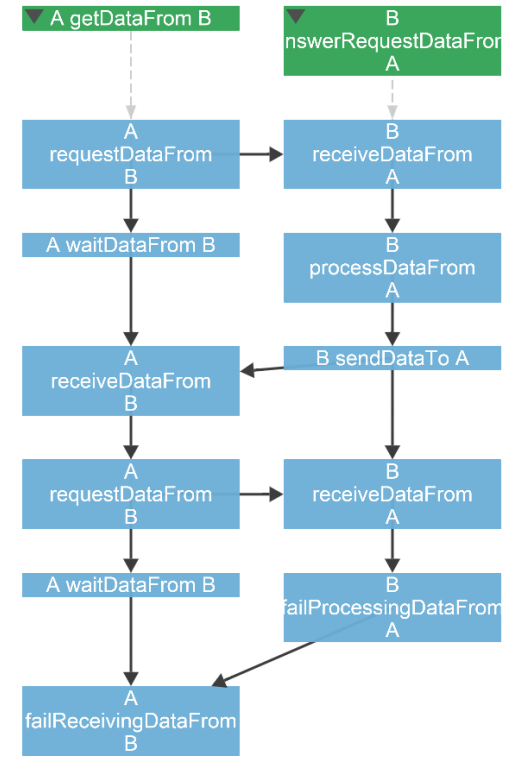

A getDataFrom B
B
nswerRequestDataFrom
A
A
requestDataFrom
B
B
receiveDataFrom
A
A waitDataFrom B
B
ailProcessingDataFrom
A
A
failReceivingDataFrom
B
A
requestDataFrom
B
B
failReceivingDataFrom
A
A waitDataFrom B
A
failReceivingDataFrom
B

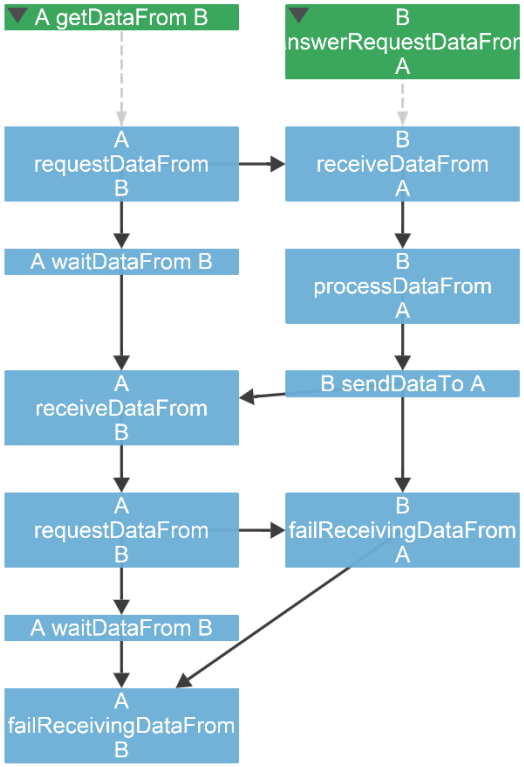

A getDataFrom B
B nswerRequestDataFrom A
A requestDataFrom B
B failReceivingDataFrom A
A waitDataFrom B
A failReceivingDataFrom B
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B ailProcessingDataFrom A
A failReceivingDataFrom B

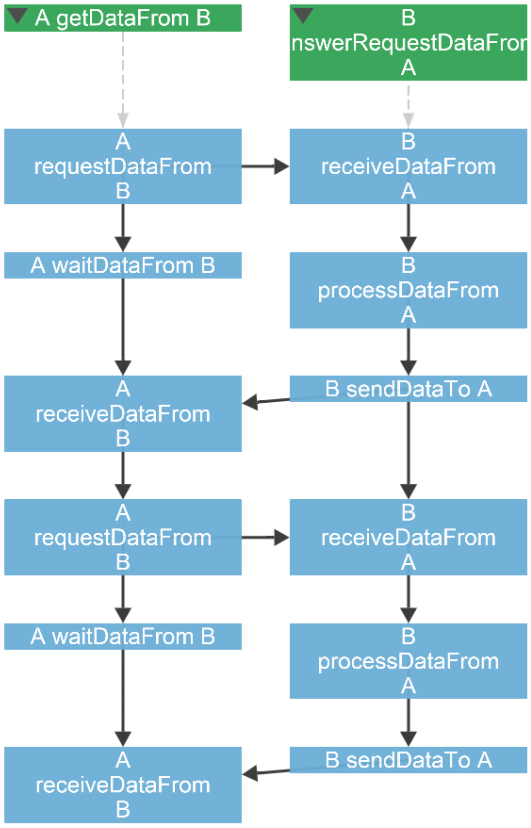

A getDataFrom B
B nswerRequestDataFrom A
A requestDataFrom B
B failReceivingDataFrom A
A waitDataFrom B
A failReceivingDataFrom B
A requestDataFrom B
B failReceivingDataFrom A
A waitDataFrom B
A failReceivingDataFrom B


A getDataFrom B
B nswerRequestDataFrom A
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B ailProcessingDataFrom A
A failReceivingDataFrom B
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B processDataFrom A
A receiveDataFrom B
B sendDataTo A


A getDataFrom B
B nswerRequestDataFrom A
A requestDataFrom B
B failReceivingDataFrom A
A waitDataFrom B
A failReceivingDataFrom B
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B processDataFrom A
A receiveDataFrom B
B sendDataTo A


A getDataFrom B
B nswerRequestDataFrom A
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B processDataFrom A
A receiveDataFrom B
B sendDataTo A
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B ailProcessingDataFrom A
A failReceivingDataFrom B


A getDataFrom B
B nswerRequestDataFrom A
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B processDataFrom A
A receiveDataFrom B
B sendDataTo A
A requestDataFrom B
B failReceivingDataFrom A
A waitDataFrom B
A failReceivingDataFrom B


A getDataFrom B
B nswerRequestDataFror A
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B processDataFrom A
A receiveDataFrom B
B sendDataTo A
A requestDataFrom B
B receiveDataFrom A
A waitDataFrom B
B processDataFrom A
A receiveDataFrom B
B sendDataTo A

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Abdelzad, Vahdat, and Fereidoon Shams Aliee. 2010. "A Method Based on Petri Nets for Identification of Aspects." In *Proceedings of Workshop on Early Aspects in AOSD*: 43–49.

Alves, Carina. 2003. "COTS-Based Requirements Engineering." In *Component-Based Software Quality*, edited by Alejandra Cechich, Mario Piattini, and Antonio Vallecillo, 21–39. Berlin: Springer.

Auguston, M. 2014. *Behavior Models for Software Architecture.* NPS-CS-14-003. Monterey, California: Naval Postgraduate School.

Auguston, M. 2009. "Monterey Phoenix, or How to Make Software Architecture Executable." In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*: 1031–1040.

Auguston, M. 2015. "Monterey Phoenix: System and Software Architecture Modeling Language." *Naval Postgraduate School, Department of Systems Engineering*, Monterey, CA.

Auguston, M. 2009. "Software Architecture Built from Behavior Models." *ACM SIGSOFT Software Engineering Notes* 34: 1–15.

Auguston, Mikhail, and Kristin Giammarco. 2015. "Firebird." Accessed July 15. http://firebird.nps.edu.

Auguston, M., K. Giammarco, C. Baldwin, J. Crump, and M. Farah-Stapelton. 2015. "Modeling and Verifying Business Processes with Monterey Phoenix." *Conference on Systems Engineering Research* 44: 345–353.

Balci, Osman. 1988. "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages." In *WSC '88 Proceedings of the 20th Conference on Winter Simulation*: 287–295.

Baldwin, W. Clifton, Brian Sauser, and Robert Cloutier. 2015. "Simulation Approaches for System of Systems: Events-Based Versus Agent-Based Modeling." In *Conference on Systems Engineering Research* 44: 363–372.

Barnes, Rory, and Richard Greenberg. 2007. "Apsidal Behavior Among Planetary Orbits: Testing the planet-planet scattering model." *The Astrophysical Journal Letters* 659: L53–L63.

Berenji, Hamid R., Yan Wang, and Abhi Saxena. 2005. “Dynamic Case-Based Reasoning in Fault Diagnosis and Prognosis.” *FUZZ-IEEE*: 845–850.

Bézivin, Jean. 2004. “In Search of a Basic Principle for Model-Driven Engineering.” *Novatica Journal, Special Issue* 5: 21–24.

Bloom, J, C Clark, C Clifford, A. Duncan, H. Khan, and M. Papantoniou. 2007. Accessed July 15, 2015. “Platform Independent Petri net Editor 2.” http://pipe2.sourceforge.net/.

Borshchev, A., and Filippov, A. 2004. “From System Dynamics and Discrete Event to Practical Agent-Based Modeling: Reasons, Techniques, Tools.” In *22nd International Conference of the System Dynamics Society*: 959–966.

Buede, Dennis. 2009. “Errors Associated with Simple versus Realistic Models.” *Computational and Mathematical Organization Theory* 15: 11–18.

Checkland, P. B. 1993. *Systems Thinking, Systems Practice.* New York: John Wiley & Sons.

Chan, Wai Kin Victor, Young-Jun Son, and Charles M. Macal. 2010. “Agent-Based Simulation Tutorial-Simulation of Emergent Behavior and Differences Between Agent-Based Simulation and Discrete-Event Simulation.” In *Winter Simulation Conference* 10: 135–150.

Clarke, E. M., and Wing, J. M. 1996. “Formal Methods: State of the Art and Future Directions.” *ACM Computing Surveys (CSUR)* 28: 626–643.

Dale, Nell, and Henry M. Walker. 1996. *Abstract Data Types: Specifications, Implementations, and Applications*. Burlington, MA: Jones & Bartlett Learning.

De Lemos, Rogério et al. 2013. “Software Engineering for Self-Adaptive Systems: A Second Research Map.” In *Software Engineering for Self-Adaptive Systems II*, edited by Rogerio de Lemos, Holger Giese, Hausi A. Muller, and Mary Shaw: 1–32. Berlin: Springer

Dolby, Julian, Mandana Vaziri, and Frank Tip. 2007. “Finding Bugs Efficiently with a SAT Solver.” In *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*: 195–204.

Eubanks, Charles F., Steven Kmenta, and Kosuke Ishii. 1997. “Advanced Failure Modes and Effects Analysis Using Behavior Modeling.” In *ASME Design Engineering Technical Conferences*: 14–17.

Graves, Henson, and Yvonne Bijan. 2011. “Using Formal Methods with SysML in Aerospace Design and Engineering.” *Annals of Mathematics and Artificial Iintelligence* 63: 53–102.

Forrester, Jay W. 1993. “System Dynamics and the Lessons of 35 Years.” In *A Systems-Based Approach to Policymaking*, edited by Kenyon B. De Greene, 199–240. New York: Springer.

Fowler, Martin. 2004. *UML Distilled: A Brief Guide to the Standard Object Modeling Language.* Boston: Addison-Wesley.

France, Robert, and Bernhard Rumpe. 2007. “Model-Driven Development of Complex Software: A Research Roadmap.” *Future of Software Engineering*: 37–54. Washington, DC: IEEE Computer Society.

French, Matthew. 2015. “Extending Model-Based Systems Engineering (MBSE) for Complex Systems.” Paper presented at the 53rd *AIAA Aerospace Science Meeting.* Kissimmee, Florida, January 5–9.

Giammarco, Kristin. 2007. “Data Centric Integration and Analysis of Information Technology Architectures.” Masters thesis, Naval Postgraduate School.

Girault, Claude, and Rüdiger Valk. 2013. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications.* Berlin: Springer.

Gore, Ross, Reynolds Paul F. Jr, Lingjia Tang, and David C. Brogan. 2007. “Explanation Exploration: Exploring Emergent Behavior.” *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*: 113–122.

Grogan, P. T., & de Weck, O. L. 2013. “An Integrated Modeling Framework for Infrastructure System-Of-Systems Simulation.” In *IEEE International Systems Conference (SysCon)*: 483–490.

Hagen, Christian, and Jeff Sorenson. 2013. “Delivering Military Software Affordability.” *Defense AT&L*: 30–34.

Hause, Matthew. 2006. “The SysML Modelling Language.” In *Fifteenth European Systems Engineering Conference*: 9.

Heath, B., Hill, R., & Ciarallo, F. 2009. “A Survey Of Agent-Based Modeling Practices (January 1998 to July 2008).” *Journal of Artificial Societies and Social Simulation*, 12(4).

Hinkelmann, Franziska, David Murrugarra, Abdul Salam Jarrah, and Reinhard Laubenbacher. 2011. “A Mathematical Framework for Agent-Based Models of Complex Biological Networks.” *Bulletin of Mathematical Biology* 73: 1583–1602.

Hürsch, Walter L., and Cristina Videira Lopes. 1995. *Separation of Concerns*. NU-CCS-95-03. Boston: Northeastern University.

Hutchinson, John, Mark Rouncefield, and Jon Whittle. 2011. "Model-Driven Engineering Practices in Industry." *Software Engineering (ICSE)* 33: 633–642.

Jackson, Daniel. 2012. *Software Abstractions: Logic, Language, and Analysis.* Cambridge, MA: MIT Press.

Jennings, Nicholas R. 2001. "An Agent-Based Approach for Building Complex Software Systems." *Communications of the ACM* 44: 35–41.

Jensen, Kurt. 2013. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*. Berlin: Springer Science & Business Media.

Johnson, Christopher W. 2006. "What are Emergent Properties and how do They Affect the Engineering of Complex Systems?" *Reliability Engineering & System Safety* 91: 1475–1481.

Kent, Stuart. 2002. "Model Driven Engineering." In *Integrated Formal Methods*, 286–298. Berlin: Springer.

Kobryn, Cris. 1999. "UML 2001: A Standardization Odyssey." *Communications of the ACM*: 29–37.

Krogstie, J. 2003. "Evaluating UML Using a Generic Quality Framework." In *UML and the Unified Process*, 1–22. Hershey, PA: IGI Global.

Krogstie, J. 2012. "Modelling Languages, Perspectives and Abstraction Mechanisms." In *Model-Based Development and Evolution of Information Systems*, 90–204. London: Springer-Verlag London.

Lambert, Jean-Luc. 1992. "A Structure to Decide the Reachability in Petri Nets." *Theoretical Computer Science* 99: 79–104.

Langford, Gary O. 2012. *Engineering Systems Integration: Theory, Metrics, and Methods.* Boca Raton, FL: CRC Press.

Martinez-Moyano, Ignacio J., and George P. Richardson. 2013. "Best Practices in System Dynamics Modeling." *System Dynamics Review* 29: 102–123.

Macal, C. M., and North, M. J. 2010. "Tutorial on Agent-Based Modeling and Simulation." *Journal of Simulation* 4: 151–162.

Maier, Mark W., and Eberhardt Rechtin. 2009. *The Art of Systems Architecting.* Boca Raton, FL: CRC Press.

Mayer, Richard. 1998. "Decidability and Complexity of Model Checking Problems for Infinite-State Systems." Ph.D. thesis, Technische Universität München.

Mellor, Stephen J. Kendal Scott, Axel UHL, Dirk Weise. 2004. *MDA Distilled: Principles of Model-Driven Architecture.* Boston: Pearson Education.

Mens, Tom, and Pieter Van Gorp. 2005. "A Taxonomy of Model Transformation." *Electronic Notes in Theoretical Computer Science* 152: 125–142.

Miyazawa, Alvaro, and Ana Cavalcanti. 2014. "Formal Refinement in SysML." In *Integrated Formal Methods*, ed. Elvira Albert and Emil Sekerinski. New York: Springer-Verlag: 155–170.

Moshirpour, Mohammad, Nariman Mani, Armin Eberlein, and Behrouz Far. 2013. "Model-Based Approach to Detect Emergent Behavior in Multi-Agent Systems." In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*: 1285–1286.

NPS. "Monterey Phoenix—Event Grammar." 2015. Accessed September 15. https://wiki.nps.edu/display/MP/Event+Grammar.

Oberndorf, Patricia, and David Carney. 1998. *A Summary of DOD COTS-Related Policies. SEI Monographs on the Use of Commercial Software in Government Systems.* Pittsburgh, PA: Carnegie Mellon Institute.

Oetsch, J, M. Prischink, J. Pührer, M. Schwengerer, and H & Tompits. 2012. "On the Small-Scope Hypothesis for Testing Answer-Set Programs." *Principles of Knowledge Representation and Reasoning* 13: 43–53.

Ogren, Ingmar. 2000. *On Principles for Model-Based Systems Engineering*. Systems Engineering 3: 38–49.

OMG. 2015. "Documents Associated with Systems Modeling Language (SysML). Version 1.4 - Beta ." *O*bject Management Group. http://www.omg.org/spec/SysML/1.4/Beta/PDF.

Penaloza, Christian, Yasushi Mae, Kenichi Ohara, and Tatsuo Arai. 2012. "Social Human Behavior Modeling for Robot Imitation Learning."In *IEEE Mechatronics and Automation*: 457–462.

Petri, A., and W. Reisig. 2008. "Petri net." *Scholarpedia* 3: 6477.

Pidd, Michael, and R. Bayer Castro. 1998. "Hierarchical Modular Modelling in Discrete Simulation." In *IEEE Simulation Conference Proceedings* 1, 383–389.

Pressman, Roger S. 2015. *Software Engineering: A Practitioner's Approach 8th Ed.* New York: McGraw-Hill.

Pullum, L. L., and Cui, X. 2012. *Techniques and issues in agent-based modeling validation.*Oakridge, TN: Oak Ridge National Laboratory.

Hamadi, Rachid, and Boualem Benatallah. 2004. “Recovery Nets: Towards Self-Adaptive Workflow Systems.” In *Web Information Systems–WISE*: 439–453. Berlin: Springer.

Raschke, Alexander. 2009. “Translation of UML 2 Activity Diagrams into Finite State Machines for Model Checking.” *Software Engineering and Advanced Applications* 35: 149–154.

Rivera Consulting. 2009. *System Architectural Modeling Interface*. Accessed August , 2015. http://eagle6modeling.riverainc.com/model.php.

Sengstacken, A.J., D.A. DeLaurentis, and M.-R. Akbarzadeh-T. 2007. “Fuzzy Logic Control for Shared-Autonomy in Automotive Swarm Environment.” In *IEEE International Conference on Systems, Man and Cybernetics*: 196–201.

Sgroi, Marco, Luciano Lavango, Yosinori Watanabe, and Alberto Sagniovanni-Vincentelli. 1999. “Synthesis of Embedded Software Using Free-choice Petri Nets.” In *ACM/IEEE Design Automation Conference*: 805–810.

Stefan, Jeff A. 2007. “Survey of Model-Based Systems Engineering (MBSE) Methodologies.” *INCOSE MBSE Focus Group* 25: 8.

Sterman, John D. 2001. “System Dynamics Modeling.” *California Management Review* 43: 8–25.

Valmari, Antti. 1998. “The State Explosion Problem.” In *Lectures on Petri Nets I: Basic Models*, 429–528. Berlin: Springer.

Weyns, D., M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. 2012. “A Survey of Formal Methods in Self-Adaptive Systems.” In *Proceedings of the Fifth International Conference on Computer Science and Software Engineering*: 67–79.

Wirth, Niklaus. 1971. “Program Development by Stepwise Refinement.” *Communications of the ACM* 14: 221–227.

Woodcock, Jim, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. “Formal Methods: Practice and Experience.” *ACM Computing Surveys (CSUR)* 41: 19.

Wulf, William A. 1980. “Abstract Data Types: A Retrospective and Prospective View.” In *Mathematical Foundations of Computer Science*, 94–112. Berlin: Springer.

Yu, Huafeng, Prachi Joshi, Jean-Pierre Talpin, Sandeep Shukla, and Shinichi Shiraishi. 2015. “The Challenge of Interoperability: Model-Based Integration for Automotive Control Software.” In *Proceedings of the 52nd Annual Design Automation Conference*: 58.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California